VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

BAKALÁŘSKÁ PRÁCE

Brno, 2019 David Ondrášek

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY
DEPARTMENT OF CONTROL AND INSTRUMENTATION

IMPLEMENTACE ALGORITMU HLUBOKÉHO UČENÍ NA EMBEDDED PLATFORMĚ
IMPLEMENTATION OF DEEP LEARNING ALGORITHM ON EMBEDDED DEVICE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **David Ondrášek**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. Karel Horák, Ph.D.**
SUPERVISOR

BRNO 2019

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní obor **Automatizační a měřicí technika**
Ústav automatizace a měřicí techniky

Student: David Ondrášek ***ID:*** 197725
Ročník: 3 ***Akademický rok:*** 2018/19

NÁZEV TÉMATU:

Implementace algoritmu hlubokého učení na embedded platformě

POKYNY PRO VYPRACOVÁNÍ:

Aplikace strojového učení v počítačovém vidění využívající prvků hlubokého učení obvykle vyžadují dlouhou dobu pro trénování mechanismu a nalezení vhodných modelů, zatímco vyhodnocení tj. klasifikace neznámého objektu vybavováním je velmi rychlé.

1. Nastudujte state-of-the-art deep learningových metod.
2. Nastudujte možnosti implementace mechanismů na různá embedded zařízení.
3. Sestavte nebo převezměte vhodný dataset pro trénovací i testovací fázi.
4. Na vybraném zařízení provedte nebo převezměte implementaci vybrané aplikace.

DOPORUČENÁ LITERATURA:

1. Goodfellow I., Bengio Y., Courville A.: Deep Learning. MIT Press, 2016. ISBN 9780262035613. (deeplearningbook.org)

2. Buduma, N., Locascio, N. Fundamentals of Deep Learning. O'Reilly Media, Inc. 2017. ISBN 9781491925614.

Termín zadání: 4.2.2019 ***Termín odevzdání:*** 20.5.2019

Vedoucí práce: Ing. Karel Horák, Ph.D.
Konzultant:

doc. Ing. Václav Jirsík, CSc.
předseda oborové rady

ABSTRAKT

Bakalářská práce se zabývá implementací inferenčního modelu, založeného na metodách hlubokého učení na embedded zařízení. V první části je provedena rešerše strojového a následně hlubokého učení a některých používaných state-of-the-art metod. V další části se práce zabývá výběrem nejlepšího vhodného hardware. Na konci kapitoly jsou podle výsledků vybrány pro implementaci Jetson Nano a Raspberry Pi. Dále je vytvořen vlastní dataset s třídami pro detekci bonbonů Maoam a na jeho základě potom vytrénován pomocí transfer learning inferenční model. Ten je potom použit při sestavení vlastní aplikace na detekci objektů, která je implementována na Jetson Nano a Raspberry Pi. Výsledky jsou vyhodnoceny a jsou naznačeny další možná budoucí vylepšení.

KLÍČOVÁ SLOVA

embedded, hluboké učení, detekce objektů, neuronová síť, transfer learning, strojové učení

ABSTRACT

This thesis deals with the implementation of inference model, based on the methods of deep learning, on embedded device. First, machine learning and deep learning methods are researched with emphasis on state-of-the-art techniques. Next, the best suitable hardware had to be selected. In the conclusion, two devices are chosen: Jetson Nano and Raspberry Pi. Then the custom dataset, consisting of three classes of candies, was created and used for training custom inference model through the transfer learning technique. Model is later used in the application, capable of object detection. Application is implemented on Jetson Nano and Raspberry Pi and then evaluated.

Keywords

Embedded, deep learning, object detection, neural network, transfer learning, machine learning

ONDRÁŠEK, David. *Implementace algoritmu hlubokého učení na embedded platformě.* Brno, 2019. Dostupné také z: https://www.vutbr.cz/studenti/zav-prace/detail/119328. Bakalářská práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky. Vedoucí práce Karel Horák.

Prohlášení autora o původnosti díla

Prohlašuji, že svou bakalářskou práci na téma Algoritmy hlubokého učení na embedded platformě jsem vypracoval samostatně pod vedením vedoucí/ho bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.
Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

.

V Brně dne: **20. května 2019** ……
David Ondrášek

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu bakalářské práce panu Ing. Karlu Horákovi, Ph.D. za odborné vedení, konzultace, trpělivost a podnětné připomínky k práci. Dále pak všem, kteří mě při vytváření práce podporovali, zvláště potom mojí přítelkyni, která přišla s návrhem na funkci aplikace.

Obsah

1 Úvod ... 1
1.1 Stručná historie strojového učení ... 3
2 Deep Learning ... 6
2.1 Model neuronu ... 6
2.1.1 Vstupní proměnné ... 6
2.1.2 Váha a bias ... 7
2.1.3 Vážená suma vnitřních hodnot ... 8
2.1.4 Aktivační funkce ... 8
2.2 Architektura ... 9
2.2.1 Konvoluční neuronové sítě (CNN) ... 11
2.2.2 Rekurentní neuronové sítě RNN ... 14
2.2.3 Transfer learning (TR) ... 14
2.2.4 Strukturované pravděpodobnostní modely ... 17
2.2.5 Auto-enkodéry SAE ... 20
2.2.6 Deep belief sítě DBN ... 21
3 Embedded zařízení ... 22
3.1 Definice ... 22
3.2 Omezení ... 22
3.3 Cloud ... 23
3.3.1 Veřejný cloud ... 23
3.3.2 Soukromý cloud ... 23
3.3.3 Komunitní cloud ... 23
3.3.4 Hybridní cloud ... 23
3.4 Procesory (CPU) ... 24
3.5 DSP ... 24
3.5.1 DSP Case study ... 25
3.6 FPGA ... 26
3.6.1 FPGA Case Study ... 27
3.7 TPU ... 28
3.7.1 Google TPU ... 29
3.7.2 VPU, APU, IPU, NPU ... 29
3.8 Grafické karty (GPU) ... 30
3.8.1 CUDA ... 31
3.8.2 Tensor Cores ... 31
3.9 AI Developer Kits ... 31
3.10 System-on-Chip (SoC) ... 34
3.10.1 SoC Case Study ... 34

3.11 Srovnání ... 37
3.11.1 Latence ... 39
3.11.2 Spotřeba ... 39
3.11.3 Flexibilita ... 39
3.11.4 Parallel Computing ... 40
3.11.5 Výkon ... 40
4 Trénování modelu ... 41
4.1 Výběr datasetu ... 41
4.1.1 Vlastní dataset ... 41
4.2 Trénink ... 44
4.2.1 Trénink v NVidia DIGITS ... 45
4.2.2 Přechod na TensorFlow ... 46
5 Implementace na zařízení ... 51
5.1 Jetson Nano ... 51
5.2 Raspberry Pi ... 55
Závěr ... 57
Literatura ... 60
Seznam symbolů, veličin a zkratek ... 66
Seznam příloh ... 67
Příloha 2 ... 68

Seznam obrázků

Obrázek 1.1: Vznik hlubokého učení [4] 4
Obrázek 1.2: Ukázka popularizace odborných prací zabývajících se hlubokým učením [6] 5
Obrázek 2.1: model perceptronu [1] 6
Obrázek 2.2: Ukázka působení biasu na výstupní funkci neuronu [7] 7
Obrázek 2.3: Porovnání aktivačních funkcí [10] 9
Obrázek 2.4: Ukázka overfitting a underfitting [11] 10
Obrázek 2.5: Ukázka fungování konvoluce [17] 12
Obrázek 2.6: Ukázka fungování pooling vrstvy [18] 12
Obrázek 2.7: Architektura jednoduché konvoluční neuronové sítě [19] 13
Obrázek 2.8: Ukázka fungování YOLO CNN[20] 13
Obrázek 2.9: Porovnání přesnosti a výpočetní náročnosti různých architektur NN [35] 17
Obrázek 2.10: Bayesova síť [60] 18
Obrázek 2.11: Markovův model [60] 18
Obrázek 2.12: RBM 19
Obrázek 2.13: Strukturovaný auto-enkodér [28] 20
Obrázek 2.14: Architektura DBN sítě [26] 21
Obrázek 3.1: Rozdělení snímku pro využití struktury více signálových procesorů [32] 25
Obrázek 3.2: Model problému šířky paměti u grafických karet [44] 30
Obrázek 3.3: Porovnání rychlosti inference různých Developer kitů [58] 33
Obrázek 3.4: Porovnání SoC [55] 36
Obrázek 3.5: Porovnání CPU, GPU, TPU: Predikce za vteřinu [43] 37
Obrázek 3.6: Porovnání CPU, GPU, TPU: výkon/spotřeba [43] 38
Obrázek 4.1: Ukázka vlastního datasetu – 1 objekt na snímek 43
Obrázek 4.2: Ukázka vlastního datasetu – více objektů ve snímku 43
Obrázek 4.3: Ukázka struktury značení objektů 44
Obrázek 4.4: Test modelu MobileNetV2 48
Obrázek 4.5: Test Modelu InceptionV2 49
Obrázek 4.6: Porovnání průběhů loss funkce 50
Obrázek 5.1: Okno v T4L při spuštění aplikace 55

Seznam tabulek

Tabulka 4.1: Kategorie vlastního datasetu 42
Tabulka 5.1: rychlosti inference 56

1 ÚVOD

Ne mnoho let zpátky pojem automatizace znamenal hlavně zavádění robotů do výroby v továrnách. Tím se docílilo výrazného zlevnění a hlavně také zpřesnění a zkvalitnění výrobků. Tato automatizace znamenala hlavně odstranění jednotvárných a jednoduše algoritmizovatelných prací a pomohla lidem posunout se dál ke kreativnějším činnostem.

V dnešní době se ale dostáváme do chvíle, kdy pomocí strojového a hlubokého učení začínáme dokazovat, že lze automatizovat i značně složitější úlohy, které nejsou jen souborem za sebou jasně jdoucích definovaných kroků, ale je za nimi hlubší úvaha a složitost.

Chceme-li přispět k budoucí automatizaci, musíme přenést tyto složitější algoritmy strojového učení na menší přenosná zařízení, abychom je mohli využít při každodenních činnostech. Tohoto přenosu se dá docílit převedením větší části výpočtů do cloudu nebo rozvinutím hlubokého učení na embedded platformách.

V této práci rozebereme strojové a hluboké učení a zaměříme se na použití těchto metod právě na embedded zařízeních. Oproti použití na počítači to znamená se hlavně zaměřit na optimalizaci algoritmů, abychom vykompenzovali obecně nižší výkon. To je svázáno s odborným výběrem hardware, který je vhodný a optimalizovaný právě pro vysoké množství paralelních výpočtů, které tyto algoritmy pro svůj správný chod potřebují.

Využití algoritmů hlubokého učení je velmi široké. Jako příklad můžeme uvést různé analýzy trhu, samořiditelná auta nebo stanovení diagnózy pacienta.

V našem případě se zaměříme na oblast počítačového vidění a budeme pomocí hluboké neuronové sítě rozpoznávat objekty v obraze. Naše implementace dokáže pomocí přeučeného modelu hluboké neuronové sítě rozpoznávat různé druhy bonbonů značky Maoam. Tato funkčnost se dá využít například při třídění bonbonů, které při výrobě vypadly z dopravníku, ale jsou jinak v pořádku a vhodné k prodeji. Jako zařízení, na kterém aplikace poběží bylo vybráno NVidia Jetson Nano. Toto zařízení je navržené pro účel použití hlubokého učení. Zároveň při datu příchodu tohoto zařízení v březnu 2019 je to novinka na trhu, která se umožňuje za malou cenu přiblížit větším specializovaným počítačům vyvíjeným pro oblast AI. Jako druhé zařízení pro implementaci aplikace bylo vybráno Raspberry Pi, jako typický zástupce obecně rozšířeného embedded zařízení.

Z historie vývoje hlubokého učení (viz část 1.1) víme, že mnohokrát nastalo období, kdy si vědci mysleli, že tato oblast je slepou uličkou a nemůže už přinést nic dalšího. Pokaždé se ale po době v řádu jednotek až desítek let objevil nový nápad, který hluboké učení vynesl znovu na výsluní a umožnil uskutečnit věci, které do té doby nebyly vůbec představitelné. Touto poslední revoluční myšlenkou

bylo právě převedení možností, které hluboké učení představuje, na přenosná zařízení, která přestala být omezována nedostatečným výkonem.

1.1 Stručná historie strojového učení

1943-1952

Walter Pitts, geniální americký matematik, vykazující geniální sklony už kolem 12. roku života a Warren McCuloch, bývalý student neurofyziologie a filozof vytvořili první matematický model neuronu. Dokázali tím, že pomocí neuronové sítě se dají počítat libovolné logické a aritmetické funkce.

Na tento model později navázali svojí práci Alan Lloyd Hodgkin a Andrew Fileding Huxley, kteří podrobněji popsali mechanismus šíření akčních potenciálů. Za tento objev získali v roce 1952 Nobelovu cenu za fyziologii a lékařství.

1957

Americký psycholog Frank Rosenblatt objevuje perceptron. Tento algoritmus využívá modelu neuronu od Waltera Pittse a Warrena McCulocha a přidává k němu způsob, jak lze učícím algoritmem určit váhový vektor. Tento algoritmus dokázal klasifikovat 2 třídy, což znamená, že na jeho výstupu mohla být logická 1 nebo 0.

Podle tohoto výzkumu také vzniknul první neuropočítač se jménem Mark I Perceptron, který dokázal rozpoznávat znaky v rozlišení 20 x 20 bodů.

1969

Vyšla kniha Perceptrons, napsaná americkými vědci Marvinem Minskym a Seymourep Papertem. Tato kniha zamítala myšlenku Franka Rosenblatta, že neuronové sítě jsou budoucností studia umělé inteligence. Toto tvrzení prezentovala na příkladu problému s vytvořením funkce XOR pomocí perceptronů, kdy tuto funkci nemůžeme vytvořit na neuronové síti pouze se vstupní vrstvou, ale potřebujeme k tomu i další skrytou vrstvu. Tento fakt byl sice známý, ale algoritmus k učení neuronových sítí s více než vstupní vrstvou nebyl v tu dobu ještě objeven.

Tato kniha přispěla k období známému jako zima umělé inteligence (z anglického AI Winter), kdy se vědci od tohoto odvětví většinou distancovali s tvrzením, že nikam dále nevede. Toto období trvalo až do 80. let.

1986

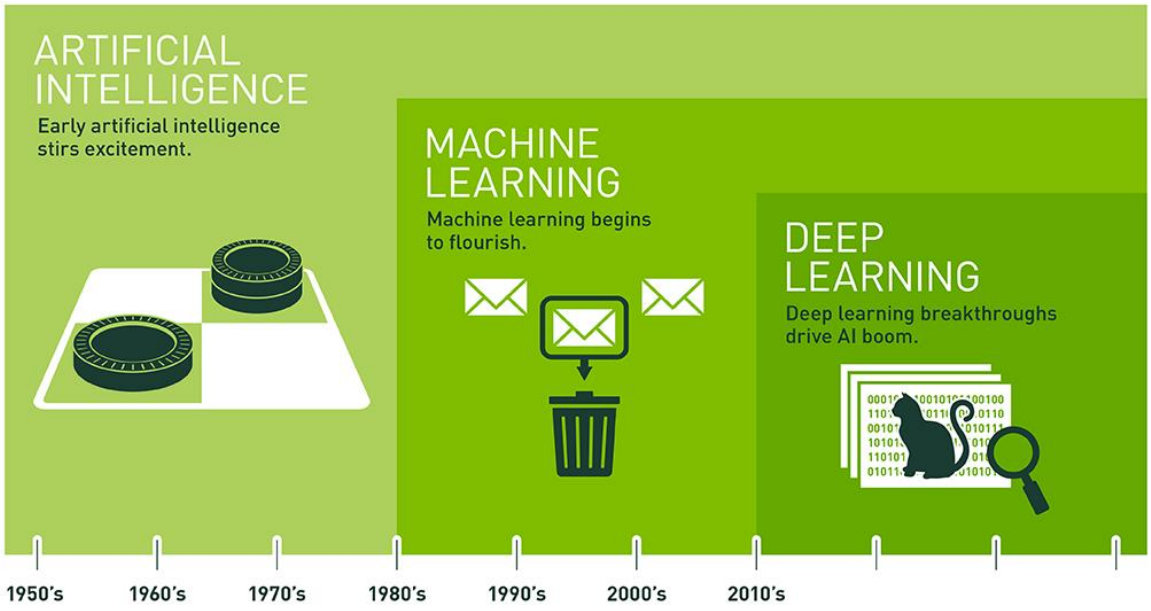
Vědci z institutu pro kognitivní vědu v University of California San Diego, Geoff Hinton, David Rumelhart a Ronald Williams zveřejnili článek Learning representations by back-propagating errors, v kterém představili učící algoritmus backpropagation i pro vícevrstvé neuronové sítě. Tento algoritmus vyřešil problém s nemožností vytvářet pomocí neuronových sítí nelineární funkce a je i dnes jedním z nejvíce používaných učících algoritmů.

1988

Francouzský vědec Yann LeCun napsal článek Gradient-Based Learning Applied to Document Recognition [2], který se zabýval rezeznáváním ručně psaných číslic. Poprvé přitom nastínil jak by se daly vytvářet konvoluční neuronové sítě (z anglického Convolutional Neural Networks). Vytvořil neuronovou síť LeNet [3].

2006

Geoff Hinton představil koncept učení neuronové sítě bez učitele (z anglického unsupervised training). Tento rok bývá považován jako první, kdy se strojové učení (machine learning) začalo vyvíjet v hluboké učení (deep learning). Obrázek 1.1 zobrazuje, jak spolu souvisí oblasti umělé inteligence, strojového učení a hlubokého učení.



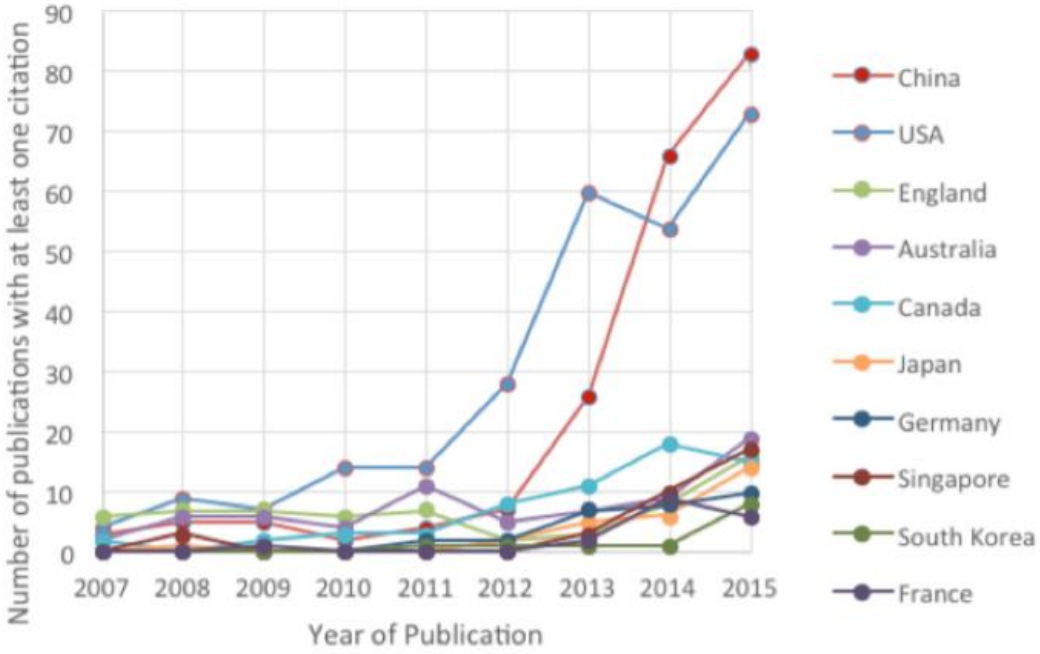
Obrázek 1.1: Vznik hlubokého učení [4]

2012

Tento rok je označován jako The Breakthrough. V tomto období už existovaly velké databáze s daty, jako je například Imagenet a některé neuronové sítě už dosahovaly velmi slibných výsledků například v rozeznávání objektů. Problém byl s velkou výpočetní náročností k trénování modelů neuronových sítí. To se velmi zlepšilo po vydání článku ImageNet Classification with Deep Covolution Neural Networks [5], který popisoval využití grafických karet k rychlejšímu trénování modelů. Zároveň byla popsána nová aktivační funkce ReLU, která lépe zabraňovala přesycení (z angl. overfitting).

Současnost

V současnosti se yvužití neuronových sítí velkým tempem zvyšuje a vychází také stále větší množství odborných článků, které se věnují právě této tématice (viz obrázek 1.2). Vznikají nové frameworky v kterých se dá s hlubokými sítěmi pracovat, větší a kvalitnější datasety a v neposlední řadě stále dochází k zvětšování výkonu hardware na kterém můžeme hluboké sítě učit. Převážně můžeme mluvit o grafických kartách, ale existují i specializované jednotky přímo určené pro učení neuronových sítí.



Obrázek 1.2: Ukázka popularizace odborných prací zabývajících se hlubokým učením [6]

2 DEEP LEARNING

2.1 Model neuronu

Perceptron se skládá ze 4 částí. První částí jsou jednotlivé vstupní proměnné $x_1 \dots x_n$, druhou částí jsou váhy vstupních proměnných a bias, neboli vychýlení, třetí částí je vážená suma vnitřních hodnot a čtvrtou aktivační funkce. Jednotlivé části můžeme vidět na obrázku 2.1, kde je vyobrazen model perceptronu. Ten se od všeobecného modelu neuronu liší tím, že na výstupu má funkci jednotkového skoku, kdežto na výstupu všeobecného modelu neuronu můžeme použít funkce různé.

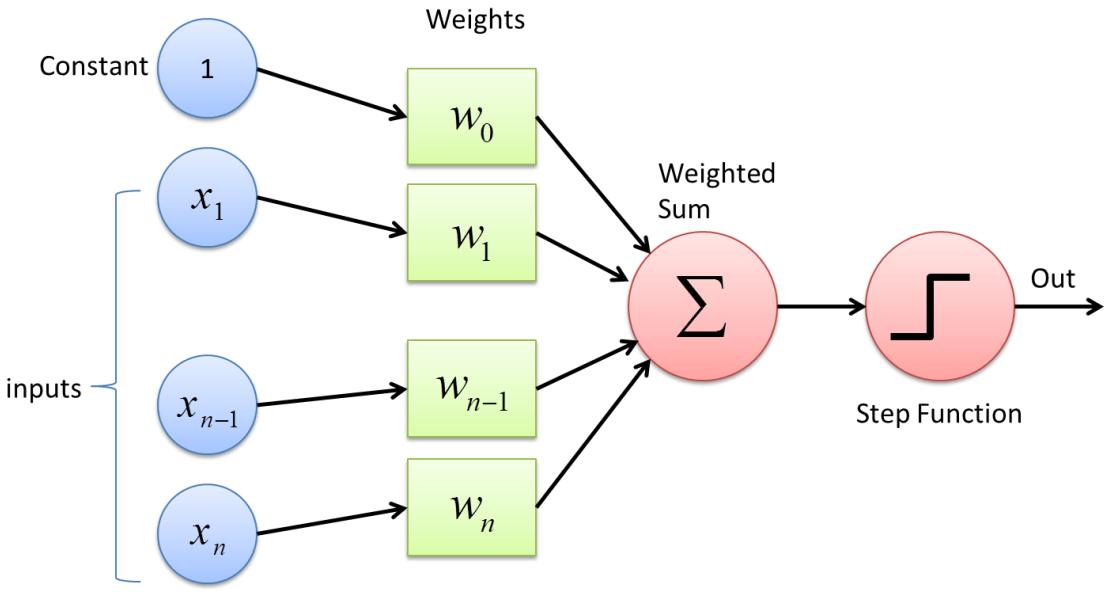
Přenosová funkce neuronu je vyjádřena:

$P: \vec{\theta} \times \vec{X}$, kde (8)

P…přenosová funkce neuronu

$\vec{\theta}$…vektor váhový vektor neuronu s váhami $\theta_1 \dots \theta_n$

$\vec{X}$…vektor vstupních veličin $x_1 \dots x_n$



Obrázek 2.1: model perceptronu [1]

2.1.1 Vstupní proměnné

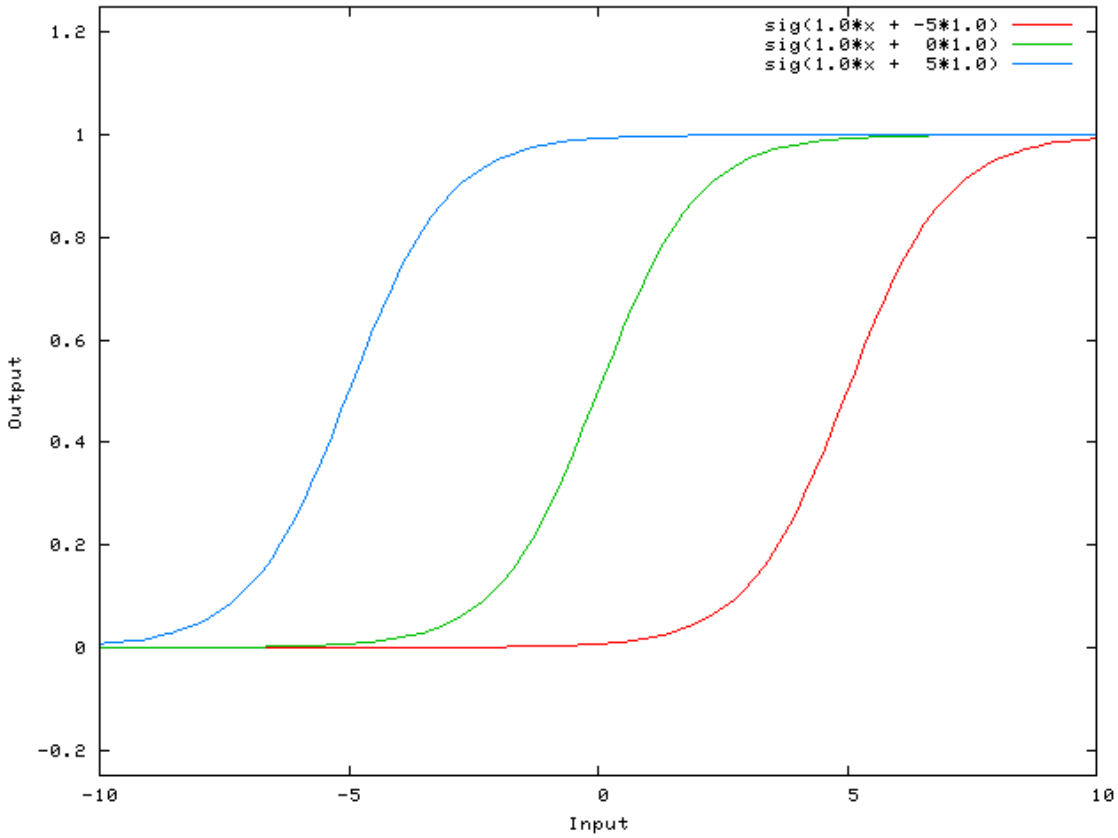
Vstupní proměnné $x_1 \dots x_n$ si můžeme představit jako jednotlivé vlastnosti, podle kterých chceme neuronovou síť nastavit a podle kterých určujeme její výstup. Jako příklad si můžeme vzít neuronovou síť, která bude podle různých parametrů

předvídat ceny domů a bytů. Vstupními proměnnými mohou být v tomto případě velikost domu, lokalita, stáří, počet pater, počet koupelen a další. Výstupem je potom samotná cena domu/bytu.

2.1.2 Váha a bias

Váhy $\theta_1 \dots \theta_n$ určují tzv. sílu jednotlivých vstupů. Můžeme si to představit na příkladu odhadování ceny domů/bytů z části 2.2.1. V tomto případě můžeme předpokládat, že největší váhu z parametrů velikost, lokalita, počet pater, počet oken, kvalita podlahy, aj. bude mít pravděpodobně velikost a lokalita, protože na konečné ceně domu/bytu se tato proměnná projevuje nejvíce. Nastavování vah je jedním z nejdůležitějších úkolů při sestavování neuronové sítě a může se provádět celou řadou různých algoritmů.

Bias si můžeme představit jako určité vychýlení, které způsobuje, že výstupní funkce neuronu se posune vůči funkci s biasem 1 doleva nebo doprava. Znázornění této funkce můžeme vidět na obrázku 2.2, kde u neuronu s jednou vstupní proměnnou a sigmoidní aktivační funkcí měníme hodnoty biasu.



Obrázek 2.2: Ukázka působení biasu na výstupní funkci neuronu [7]

2.1.3 Vážená suma vnitřních hodnot

Sčítá váhy $\sum x_i \,.\, \theta_j$ všech vstupů a podle výsledného součtu nastavuje výstupní funkci.

2.1.4 Aktivační funkce

Nastavuje výstup neuronu podle určité funkce. Obvykle se využívají funkce, které na svém výstupu mají reálné číslo z intervalu $< 0; 1 >$. Takovým příkladem mohou být funkce jako skoková funkce, kterou využívá perceptron, sigmoidní funkce, funkce tanh, ReLU, PReLU a další.

2.1.4.1 Sigmoidní funkce

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Je jednou ze základních a nejvíce využívaných aktivačních funkcích v dnešních aplikacích, i když se už od ní začíná upouštět ve prospěch modernějších funkcí, jako je například ReLU. I tak má ale široké možnosti využití. Jeví se například jako ideální aktivační funkcí pro jednodušší klasifikátory [9].

Výhodou sigmoidní funkce je její samotná podstata, kdy jejím výstupem je přímo hodnota z intervalu $< 0; 1 >$, takže už tuto hodnotu nemusíme dále upravovat.

Nevýhodou je její příkrost okolo výstupní hodnoty 0,5, což znamená že hodnoty v této oblasti se budou měnit velmi rychle, kdežto hodnoty okolo jedničky a nuly se budou měnit pomalu. Také v těchto bodech nastává problém zmenšování strmosti derivací a tím pádem i zpomalování učení neuronové sítě.

2.1.4.2 Funkce tanh

$$f(x) = \frac{2}{1 + e^{-2x}} - 1$$

Tato funkce má oproti sigmoidní funkci rozsah výstupních hodnot od -1 do 1. Podléhá taky méně rychlé změně výstupních hodnot ve středním pásmu funkce. Jinak je ale sigmoidní funkci velmi podobná a stejně jako ona podléhá i zmenšování gradientu v bodech blížících se 1 a -1.

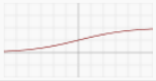
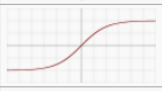
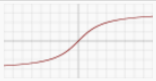
2.1.4.3 ReLU

V moderních aplikacích a pracích asi nejvíce využívaná aktivační funkce ReLU se od předchozích podstatně liší. V záporné části osy x má nulovou výstupní hodnotu a v kladné části potom lineární přímku.

Hlavní výhodou funkce ReLU je její menší výpočetní náročnost a také fakt, že neurony, které mají jako výstup nulu se ve své podstatě vůbec nepodílí na výpočtech v dalších vrstvách a tím dochází ke zrychlení sítě a jejímu odlehčení

Nevýhodou při používání nijak nemodifikované ReLU může být takzvané „umírání neuronů“. K tomu dochází z důvodu nulové derivace v záporné části osy x, takže neuron se potom dále neučí a neprovádí další výpočty. Tomuto problému se dá vyhnout nahrazením čistě horizontální linie linií s funkcí například y = 0,01x.

V obrázku 2.3 můžeme vidět průběhy výše zmíněných funkcí společně s dalšími využívanými aktivačními funkcemi

Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1-f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2+1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU) [2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) [3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1+e^x)$	$f'(x) = \frac{1}{1+e^{-x}}$

Obrázek 2.3: Porovnání aktivačních funkcí [10]

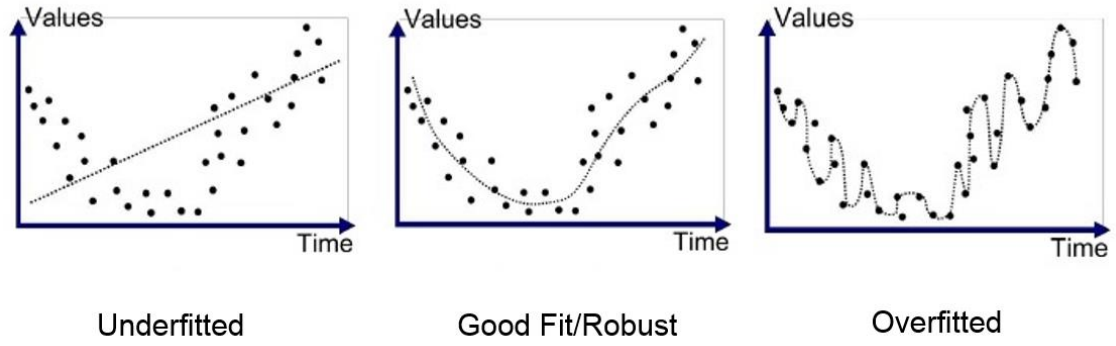
2.2 Architektura

Pod samotnou architekturou neuronových sítí myslíme vlastně neurony popsané v kapitole 2.1 spojené v několika vrstvách. V první vrstvě najdeme vstupní proměnné a v poslední je výstup neuronové sítě. Vrstvy 2 až n-1 potom nazýváme skrytými vrstvami. Jejich počet a velikost je závislý na složitosti zpracovávané informace a na tom, jak složitá je procedura, kterou musí neuronová síť zvládnout.

Příkladem může být již dříve zmíněný problém s funkcí XOR, kterou nemůžeme simulovat na neuronové síti bez skryté vrstvy.

Overfitting a underfitting

Počet vrstev i neuronů ve vrstvách musí být správně vybalancován, aby nedocházelo k tzv. „overfitting" nebo "underfitting" problémům. Overfitting znamená přílišné lpění neuronové sítě na trénovacím datasetu a následné větší chybě na testovacím datasetu. Jinak řečeno síť je přetrénovaná a až příliš se přizpůsobila hodnotám na kterých byla vytrénována. Naopak underfitting znamená, že se síť vytrénovala pouze na omezeném trénovacím datasetu a na testovacím datasetu kvůli tomu vykazuje velkou chybu. Ukázku můžeme vidět na obrázku 2.4.



Obrázek 2.4: Ukázka overfitting a underfitting [11]

Strojové učení a hluboké učení

Machine learning přechází v deep learning v tu chvíli, kdy nechceme pouze predikovat možnost s nejvyšší pravděpodobností podle předchozích výsledků, ale také vyvozovat nové závěry. Hluboké učení (deep learning) tak pracuje s vyšší abstrakcí dat, kdy výsledek zdánlivě ani nemusí se vstupními daty nijak souviset. Do principu hluboké neuronové sítě ani vlastně nemusíme vidět, protože se může stát, že po vytrénování se může začít chovat jako blackbox a nebudeme vědět co se jednotlivé vrstvy vlastně naučily. To může být občas v reálných aplikacích na obtíž, a obzvláště v aplikacích na embedded platformě. Znamená to totiž, že neumíme přesně popsat, jak se algoritmus zachová ve všech možných situacích a tím pádem je verifikace dané aplikace složitější a u takových zařízení jako jsou lékařské přístroje ani není potom většinou možná. Deep learning také umožňuje sám si hledat parametry (features) z dodaných vstupních dat [12].

Jako příklad může sloužit projekt AlphaGo od firmy Google [47]. Je to program, který díky využití hlubokého učení dokázal porazit ty nejlepší velmistry ve hře Go, o které se dlouho tvrdilo, že je tak složitá, že žádný program nemůže člověka porazit. Tento program nefunguje na principu pouze strojového učení.

Nehledá pouze ten nejlepší tah, který se statisticky osvědčil. Místo toho dokáže sám hledat vlastní strategie, které se nejlépe hodí pro daný okamžik a daného soupeře.

Můžeme tak říct, že hluboké učení je součást strojového učení a používá jeho metody, ale vylepšuje ho tak, že se snaží i interpretovaným výsledkům porozumět a podle toho se zařídit. V této práci se budeme zabývat dále právě deep learningovými (DL) metodami.

Mezi ty patří:

- Konvoluční neuronové sítě (CNN – Convolutional Neural Networks),
- Rekurentní neuronové sítě (RNN – Recurrent Neural Networks),
- Transfer Learning (TR),
- Strukturované pravděpodobnostní modely (Structual Probabilistic Models),
- Auto-enkodéry (SAE – Stacked Auto-encoder),
- Deep belief sítě (DBN – Deep Belief Networks). [13][60]

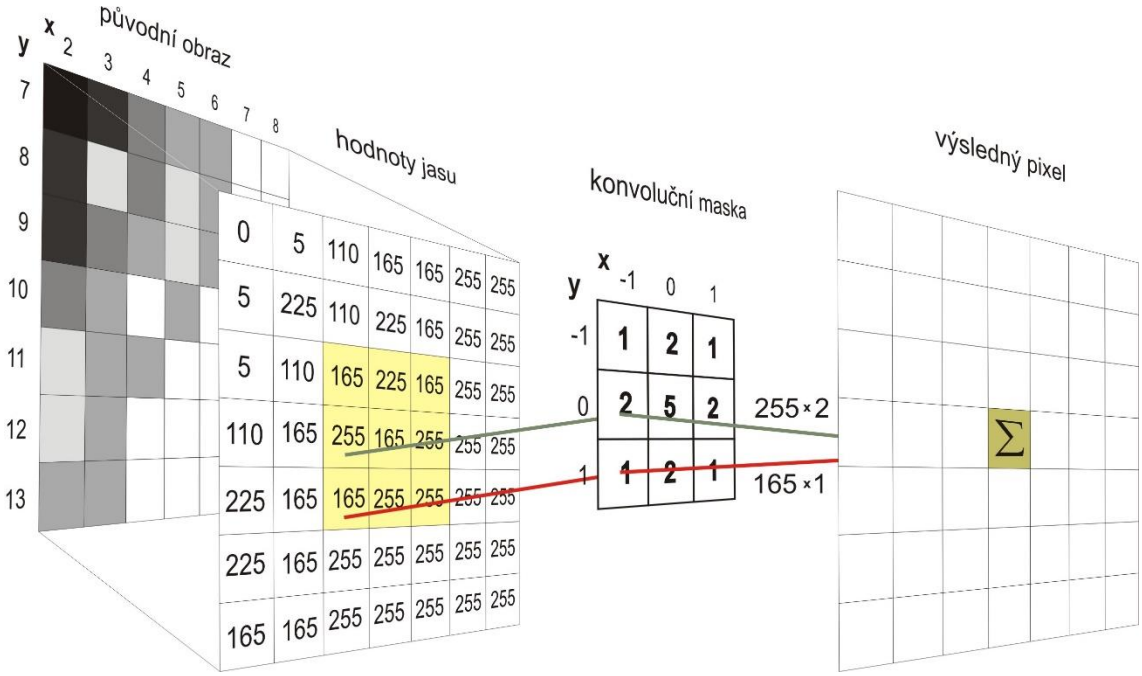
2.2.1 Konvoluční neuronové sítě (CNN)

Jsou v dnešní době asi nejvíc využívanou architekturou hlubokého učení. Jsou velmi účinné na práci s obrazy, takže se hojně používají v oblastech počítačového vidění (z angl. computer vision).

Princip konvolučních neuronových sítí spočívá v konvoluci vstupních dat přes takzvané filtry, zpracování pomocí aktivační funkce (nejčastěji ReLU) a zmenšení takto filtrovaných dat přes pooling vrstvy. Těchto vrstev může být v celé síti mnoho. Nakonec dochází ke klasifikaci výstupu z poslední vrstvy.

2.2.1.1 Konvoluce

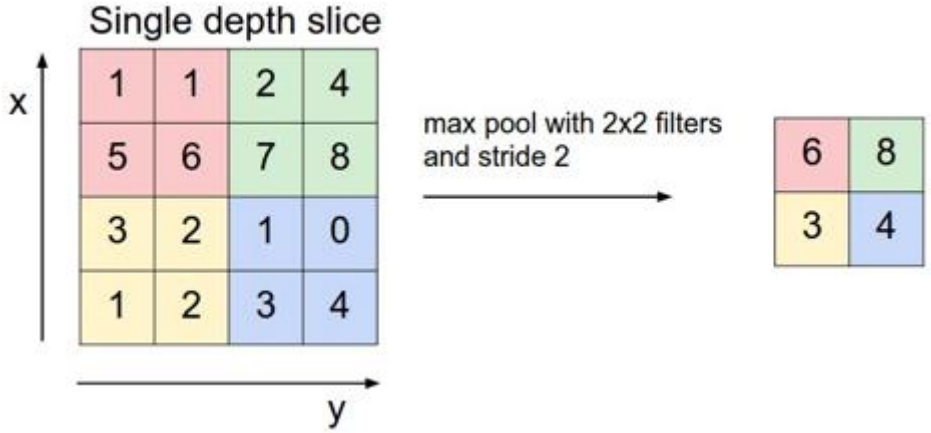
Je to matematická operace, při které ze vstupní matice dat pomocí jiné menší matice, kterou značíme jako filtr získáme příznak, pro který byl filtr navržen. Jestliže máme například filtr o rozměrech 3x3, tak při zakrytí 9 pixelů u vstupního obrázku dat dochází ke konvoluci tak, že hodnotu každého pixelu vstupních dat vynásobíme hodnotou u filtru na místě, kde se hodnoty překrývají a potom vynásobené hodnoty sečteme dohromady. Tento součet ještě prochází aktivační funkcí. Filtr se potom dále posouvá dál a provede konvoluci znovu postupně pro celý obraz. Vzniká tím další vrstva příznaků (feature map). Počet hodnot, o které se filtr posouvá dále se nazývá „stride“. Na obrázku 2.5 je příklad fungování konvoluce při zpracování obrazu.



Obrázek 2.5: Ukázka fungování konvoluce [17]

2.2.1.2 Pooling

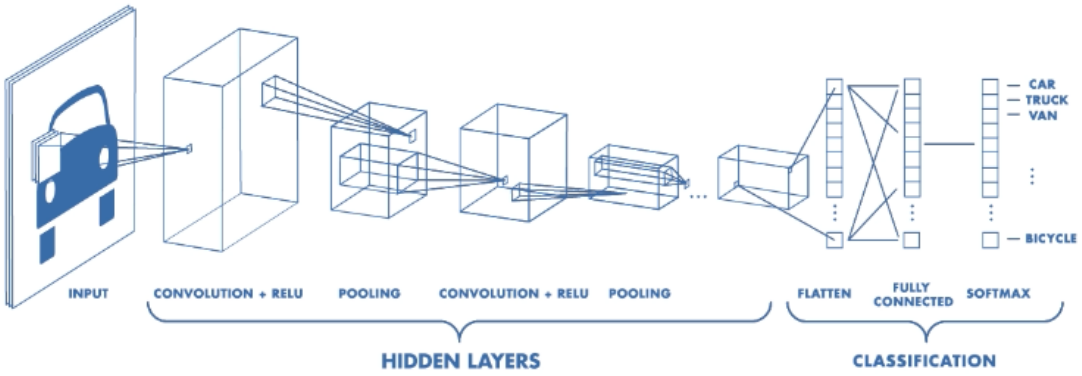
Je to způsob jak zredukovat předchozí vrstvu, abychom mohli snížit výkon potřebný pro trénování sítě a také aby nedocházelo k overfitting. Probíhá tak, že z předchozí matice vybereme několik stejně velkých submatic a z nich vždy vybereme určitou hodnotu, kterou zapíšeme do další vrstvy. Většinou se používá strategie výběru nejvyšší hodnoty.



Obrázek 2.6: Ukázka fungování pooling vrstvy [18]

2.2.1.3 Architektura CNN

Každou další konvolucí získáváme novou mapu příznaků. Těmito příznaky mohou být například jednoduché geometrické tvary v první vrstvě sítě a složité prostorové struktury v poslední vrstvě. CNN tímto způsobem klasifikuje v obrazech objekty.



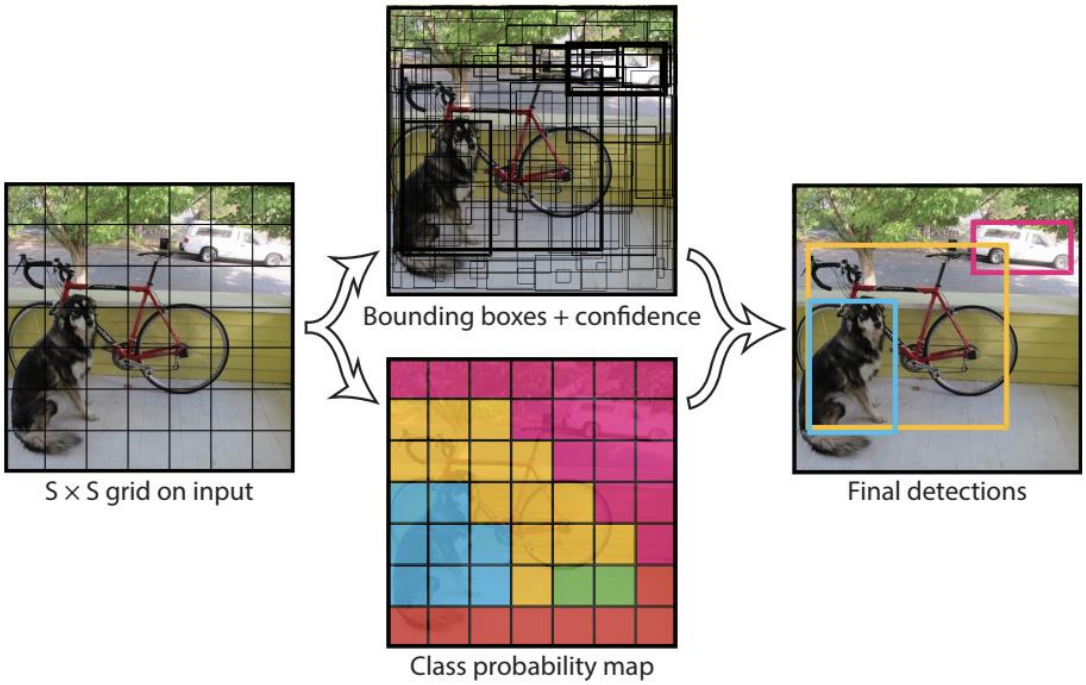
Obrázek 2.7: Architektura jednoduché konvoluční neuronové sítě [19]

2.2.1.4 State-of-the-art CNN algoritmy

V této části uvedeme některé vyspělé inferenční algoritmy pro konvoluční neuronové sítě pro detekci objektů.

YOLO

Tento algoritmus nehledá objekty v celém obraze, ale místo toho si nejdřív pomocí jednodušší CNN rozdělí obraz na menší boxy a v těch kde je nejpravděpodobnější možnost výskytu potom detekuje objekt.



Obrázek 2.8: Ukázka fungování YOLO CNN [20]

R-CNN
Tento algoritmus si celý obraz rozdělí na 2000 oblastí a pomocí algoritmu selective search [21] a SVM (support vector machine) najde oblasti kde se s největší podobností nachází objekt který chceme klasifikovat.

Fast R-CNN
Zrychlený algoritmus R-CNN. Oblasti, které prohledává selective search se zmenší pomocí první CNN, která obraz předpřipraví jednou konvoluční vrstvou a následným poolingem.

SDD
SDD, neboli také Single Shot Detector je moderní algoritmus, který vyniká nad ostatními svojí vyšší přesností. Rychlost detekce je vysoká, ale zaostává například za Faster R-CNN [62]. SDD vytváří pro klasifikaci objektů oproti předchozím více ohraničujících boxů, které vznikají v některých konvolučních vrstvách. Počet boxů závisí na počtu možných klasifikovaných tříd. Pro každý box rovnou vzniká predikce výsledku pro každou třídu. V poslední vrstvě jsou potom tyto predikce vyhodnoceny.

2.2.2 Rekurentní neuronové sítě RNN

Můžeme si je představit jako neuronovou síť s jistou pamětí. Posíláme do nich totiž sekvenci dat, u nichž je důležité jejich pořadí. Příkladem je rozpoznávání řeči, kde jsou RNN nejvíce využívané. Slovo je totiž mnohem lépe rozeznatelné, když víme jaká další slova mu předcházela. RNN používá například pro rozpoznávání řeči i operační systém Android. Další možné využití je v kombinací s CNN, kdy RNN dokáže popsat co jsou zač detekované objekty.

Pokud budeme pokračovat v analogii rozpoznávání řeči, tak vstupními daty je věta. Z té vezmeme první slovo a pomocí jiné neuronové sítě určíme jeho význam. Parametry tohoto slova potom slouží jako první vrstva RNN a další slova jako další vrstvy. Určování každého dalšího slova ve větě je tedy ovlivněno slovy předchozími.

2.2.3 Transfer learning (TR)

Transfer learning spadá do oblasti Representation learning. Jsou to metody, díky kterým je možné vylepšit reprezentaci dat tak, aby bylo snazší další trénování modelu. Když vezmeme například v potaz, že poslední vrstvou NN je klasifikátor 1000 tříd, tak si můžeme představit, jak vhodná reprezentace vah v prvních vrstvách pomůže rychlosti přeučení NN na 1001 tříd díky správnému rozpoznání

příznaků jako jsou některé jednoduché tvary v prvních vrstvách. Potom při přetrénování sítě můžeme odstranit váhy v poslední klasifikační vrstvě a vytrénovat je znovu s novou definicí tříd.

Transfer learning znamená, že pomocí modelu jiné, už vytrénované NN vytrénujeme naši síť, která bude například klasifikovat jiné objekty. Tento proces je časově kratší než proces trénování celé sítě znovu od začátku [22] [61].

K dispozici jsou kvalitní předtrénované modely jako například modely ze stránky https://modeldepot.io, http://pretrained.ml nebo také předtrénované modely od tvůrců frameworků TensorFlow a Caffe. Tvůrci většinou nazývají kolekci svých předtrénovaných modelů Model Zoo. K trénovaní těchto modelů bylo využito velkých databází jako například ILSVRC. Kdybychom měli trénovat síť stejně velikým objemem dat, bylo by to výpočetně velmi náročné. Dostupné modely mají rozdílné architektury a jsou tak využitelné pro rozdílné situace. Jako známé architektury modelů můžeme uvést například AlexNet, GoogleLeNet/Inception, MobileNet nebo ResNet.

2.2.3.1 AlexNet

AlexNet je jedna z nejznámějších architektur NN. V roce 2012 tato architektura vyhrála ImageNet LSVRC-2012. Model byl vytrénován pomocí datasetu ILSVRC a jeho vstupem je obraz o rozlišení 256x256 pixelů. Upravením této architektury později vznikla celá řada dalších. Jako první přišla s aktivační funkcí ReLU a s vícenásobnými konvolučními vrstvami. Navzdory tomu, že byla tato síť vytrénovaná obrovským datasetem, je poměrně náchylná k problému overfitting. K vyhnutí se tomuto problému se doporučuje například místo vstupu obrazu o rozlišení 256x256 použít oříznutý obraz o velikosti 224x224 pixelů [49].

2.2.3.2 GoogleLeNet/Inception

GoogleLeNet je vítěz ILSVRC z roku 2014, kdy tuto soutěž tento model vyhrál s chybou pouze 6,67 %. GoogleLeNet je pouze názvem jednoho konkrétního modelu použitého při soutěži, odvozeného od architektury Inception. Tato architektura obsahuje 22 vrstev a její hlavní výhodou je její menší velikost a větší rychlost při inference oproti větším sítím. To je docíleno tím, že má méně parametrů. Oproti NN AlexNet, která jich má 60 milionů, jich GoogleLeNet obsahuje pouze 4 miliony. Při implementaci je tato velikost znát, takže model AlexNet může mít okolo 200MB, kdežto GoogleLeNet 50MB. Celá tato síť byla navržena s důrazem na využití v menších a méně výkonných přenosných zařízeních, proto je vhodná i pro implementaci do embedded zařízení.

Vstupem je obraz o rozlišení 224x224 pixelů a novinkou, která pomohla k snížení počtu parametrů bylo zavedení inception modulů. Tento model dokáže provést několik operací typu konvoluce nebo pooling najednou s použitím

takzvané bottleneck vrstvy za minimálního počtu násobení. Dochází tak k výraznému snížení nutné výpočetní síly [50].

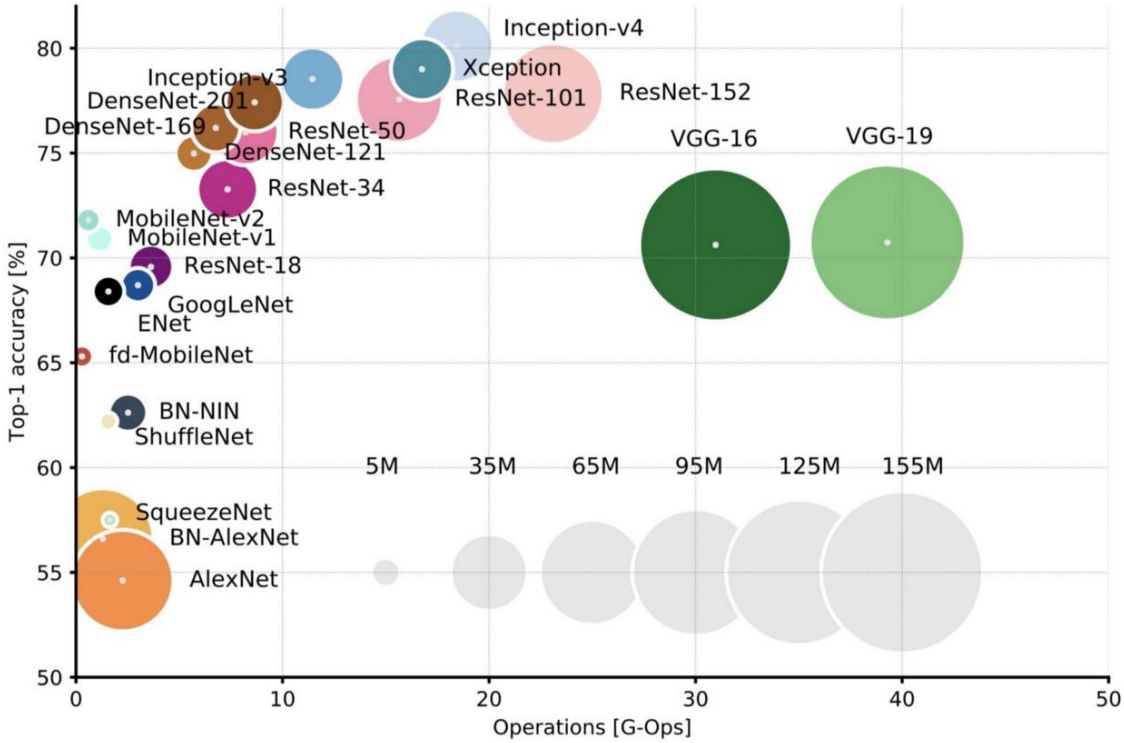
2.2.3.3 MobileNet

MobileNet je další architektura vyvinutá Googlem s účelem pro využití v přenosných a embedded zařízení. Existuje více variant MobileNet podle velikostí vytrénovaného modelu. Článek [51] pojednává o přetrénování modelu MobileNet na rozpoznávání silnice. Z porovnání vyšlo, že přesnost rozpoznávání objektu s modelem MobileNet 1.0 (95,5 %) je srovnatelná s přesností rozpoznávání objektu modelem InceptionV3 (95,9 %). Markantní rozdíl byl ale v rychlosti a velikosti modelu, kdy při chodu na grafické kartě NVIDIA 960m docházelo k rychlosti zpracování 19 snímků za sekundu u Inception a 135 u MobileNet. Velikost modelu Inception byla 89MB a velikost MobileNet pouhých 17MB. U testu odlehčené verze MobileNet 0.25 se sice snížila přesnosta na 89,2 %, ale rychlost se zvýšila až na 450 snímků za sekundu při modelu zabírajícím pouhých 930kB paměti.

Velká rychlost a malá velikost je způsobena odlehčenou architekturou sítě díky využití speciální konvoluce nazývané Depthwise Separable Convolution a následné Pointwise Convolution. Tyto formy konvoluce jsou oproti standartní extrémně účinné, ale za cenu pouhého filtrování vstupních kanálů a ne jejich míšení a vytváření nových parametrů [52].

2.2.3.4 ResNet

Tento model vyhrál v roce 2018 soutěž ILSVRC-2015 s chybou pouze 3,57 %. Tato chyba už je menší než chyba rozpoznávání objektů člověkem. Tvůrci tohoto modelu řešili problém zvyšující se chyby při rozpoznávání při přidávání více vrstev do sítě. Ten vyřešili vložením reziduálních bloků (residual blocks). V principu jde o to, že v architektuře sítě se mezi 2-3 vrstvami nacházejí jakési zkratky mezi jednotlivými vrstvami, které přidávají do dalších částí původní parametr. Tím se dá snížit komplexnost celé sítě a předejít tak této chybě. Díky těmto blokům obsahuje tato síť 152 vrstev [53].



Obrázek 2.9: Porovnání přesnosti a výpočetní náročnosti různých architektur NN [35]

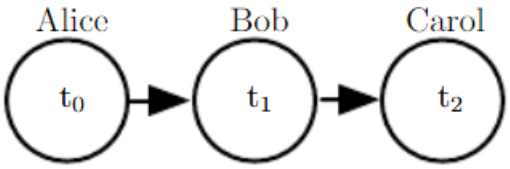
2.2.4 Strukturované pravděpodobnostní modely

Tyto modely graficky vyjadřují pravděpodobnosti jednotlivých událostí a jejich závislosti. Dělí se na tří hlavní kategorie: orientované (directed), kde je hlavním představitelem Bayesova síť, neorientované (undirected) s představitelem Markovovým modelem a energetické s Omezeným Boltzmannovým strojem. V DL jsou tyto modely nejvíce využívány pro předtrénování sítě s ohledem na pravděpodobnostní výskyt některých příznaků.

2.2.4.1 Bayesova síť

Tento orientovaný model vyjadřuje závislost jedné události na druhé pouze v jednom směru, tzn. orientovaně. Představme si, že probíhá štafetový závod s běžci Alice, Bob a Carol. Konečný čas doběhnutí je pro Boba závislý na tom, jak rychle doběhne Alice a pro Carol jak doběhne Bob. Pro Carol existuje nepřímá závislost na tom, jak doběhne Alice [60].

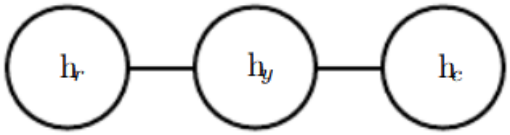
Model je použitý například při tvorbě Bayesovy konvoluční neuronové sítě, která dokáže částečně negovat vlivy, které působí datasety s pouze malým množstvím pozitivních klasifikovaných výsledků. Jedná se například o úlohu detekce rakoviny, kdy je v datasetu identifikován pozitivní výsledek pouze například v poměru 1000:1 negativních ku pozitivním výsledkům (pozitivním výsledkem je míněna přítomnost rakoviny v tkáni).



Obrázek 2.10: Bayesova síť [60]

2.2.4.2 Markovův model

Na rozdíl od Bayesova modelu u Markovova modelu neurčujeme směr, v kterém se události šíří a ovlivňují další události. Místo toho modelujeme, které události na sebe mohou mít vliv a ty potom spojujeme pouze čárou bez šipky jako v obrázku 2.11 [60].



Obrázek 2.11: Markovův model [60]

2.2.4.3 Omezený Boltzmannův stroj (RBM – Restricted Boltzmann Machine)

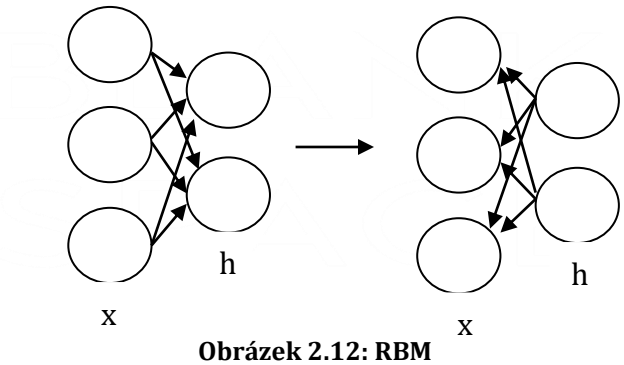
RBM je příkladem neuronové sítě, jejíž učení probíhá bez učitele (z angl. unsupervised learning). To znamená, že poskytujeme vstupní data X, ale nedefinujeme žádný chtěný výstup Y. Úlohou v unsupervised learning je vyhledání nejvíc smysluplných parametrů, které definují vstup X.

V případě RBM se neuronová síť skládá pouze z dvou vrstev: jedné vstupní a jedné skryté. Hledání správných vah potom probíhá pomocí hledání tzv. minimální energie podle následující rovnice:

$E(x, h) = a^T x - b^T h - a^T W h,$
kde x je vektor vstupních proměnných,
h je vektor proměnných ze skryté vrstvy,
a, b jsou bias vstupní a skryté vrstvy,
W je váhová matice.

Při trénování RBM se nejprve zvolí náhodné váhy neuronů. Poté proběhne klasické násobení vstupů vahami, součet a průchod aktivační funkcí. Následně se chod neuronové sítě „obrátí“ a hodnoty ve skryté vrstvě se posílají stejným způsobem zpátky do vstupní vrstvy a vzniká takzvaná rekonstrukce. Rozdíl hodnot z rekonstruované vrstvy a ze skryté vrstvy nám potom určí energii, kterou se snažíme postupně minimalizovat.

V průběhu dojde k průniku dvou pravděpodobností vypočítaných při dopředném a zpětném pohybu po neuronové síti (backward pass a forward pass). [23]



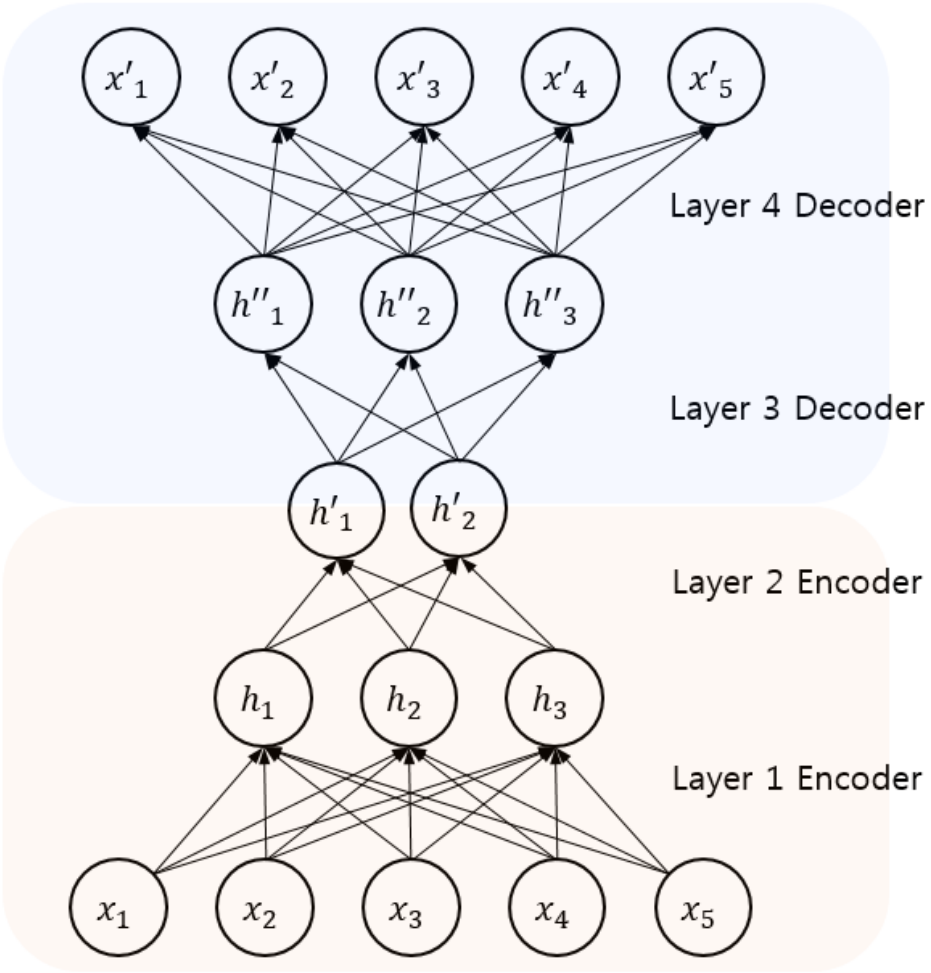
Obrázek 2.12: RBM

2.2.5 Auto-enkodéry SAE

Auto-enkodér je architektura neuronové sítě o třech vrstvách, která redukuje počet vstupních proměnných a odstraňuje tak parazitní jevy, jako je například šum. První vrstva se nazývá kodér a přes charakteristickou rovnici redukuje dimenzi vstupních proměnných. Tento výstup se potom promítne v kódové vrstvě a znovu se dekóduje do vrstvy dekodéru. Tímto procesem vlastně rekonstruujeme původní data, ale odstraňujeme z nich nežádoucí šum.

Strukturovaný auto-enkodér (SAE) je potom síť sestavená z více takových auto-enkodérů. Trénuje se postupně greedy způsobem stejně jako RBM v DBN. Nejdříve se tedy vytrénují všechny auto-enkodéry zvlášť a potom se složí do vnořené struktury [27].

V obrázku č. 2.13 vidíme 2 do sebe vnořené auto-enkodéry



Obrázek 2.13: Strukturovaný auto-enkodér [28]

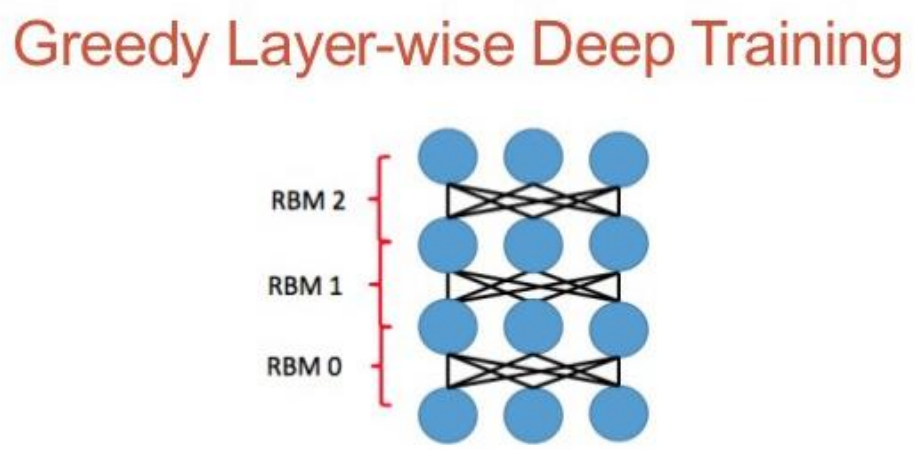
2.2.6 Deep belief sítě DBN

Patří společně s CNN mezi hojně používané architektury dnešních hlubokých neuronových sítí.

Stavebním prvkem DBN jsou RBM popsané výše, které jsou seřazeny za sebou. Postupně dochází k učení jednotlivých RBM a po dokončení tohoto procesu se použije skrytá vrstva RBM jako vstupní vrstva pro další RBM. Ze staré vstupní vrstvy se stává Bayesova síť. Každá další vrstva bude mít vyšší stupeň vjemů z předchozích vrstev a bude vidět větší celky. Výhodou je, že způsob trénování této sítě je tzv. greedy, což znamená se trénují malé celky RBM a potom se mohou skládat dohromady. [25]

Po vytrénování, kdy DBN vhodně zvolí váhy v celé síti se potom pomocí učení s učitelem (supervised learning) například. backpropagation algoritmem vytvoří samotný využitelný model sítě pro chtěné využití.

DBN je příkladem složitější neuronové sítě uzpůsobené pro unsupervised learning a je dnes hojně využívána. NASA například tento princip používá pro zpracování obrazů s vysokým rozlišením z jejich satelitů [24]



Obrázek 2.14: Architektura DBN sítě [26]

3 EMBEDDED ZAŘÍZENÍ

3.1 Definice

Abychom mohli dále pracovat s pojmem embedded zařízení, je třeba si nejdřív definovat co to vlastně je.

Techopedia definuje embedded device jako vysoce specializované zařízení, které slouží převážně jen k několika málo specifickým účelům. Zároveň zmiňuje, že tato zařízení můžou být součástí většího systému, jakým jsou například chytré hodinky se zabudovaným měřičem tepu (čili embedded zařízením) [14].

Na stránce embedded.com, která se zabývá výhradně embedded zařízeními se tyto definice různí. Jedna verze je, že to je počítač, který nevykonává funkce jako počítač. Druhá verze říká, že to musí být systém, jehož hardware musí být optimalizován pro hlavní aplikaci embedded zařízení [15].

Dohromady ale vychází najevo, že hranice mezi embedded a „normálním“ zařízením se stále zužuje a například mobilní telefony nemůžeme v dnešní době zařadit ani do jedné kategorie. Proto se v případě této práce uspokojíme s volně interpretovanou definicí, která říká, že embedded zařízením může být jakékoli zařízení, které slouží především k několika předem definovaným činnostem.

3.2 Omezení

Jako největší omezení pro běh deep learning algoritmů na embedded zařízeních se jeví omezený výkon, který je ve většině případů zapříčiněný právě přenosností, tzn. velikostí zařízení a s tou spojenou nemožností například lepšího chlazení nebo napájení pro větší výkon. Tento problém se částečně řeší pomocí trénování a nastavování neuronových sítí na výkonnějším zařízení a potom následném nahrání hotové neuronové sítě na embedded zařízení, přičemž se síť už dále netrénuje, ale na zařízení probíhá pouze inferenční fáze. I tak je ale výkonová náročnost při chodu již hotové sítě velká. Velké sítě se také musí zmenšit a optimalizovat.

Dalším omezením je samotné nahrávání na některé mikročipy. Překlad do strojových instrukcí mikročipu není vždy správně optimalizován a tím dochází k dalším ztrátám výpočetního výkonu.

V potaz také musíme vzít, že embedded zařízení bývá často součástí většího celku a přístup k němu nemusí být jednoduchý. To znamená, že nahrávání nové vylepšené neuronové sítě může být složité i z hlediska přístupu k zařízení. Nahrávání také bude z pravidla trvat déle než pouhé zkopírování do počítače.

3.3 Cloud

Cloud může nabídnout mnohá řešení problémů s velikostí výkonu, úložného místa nebo aktualizací neuronové sítě novými daty. Zároveň s tím ale přináší problémy jiné. Nutnost internetového připojení, závislost na cizích serverech a s tím související přistoupení na cizí obchodní podmínky, možnost zneužití našich dat nebo špatnou podporu providera.

Hrozbou ovlivňování našich dat či omezení slíbeného výkonu vinou špatného poskytovatele cloudu se zabývá například článek SafetyNets: Verifiable Execution of Deep Neural Networks on an Untrusted Cloud, v kterém autoři navrhují SafetyNets protokol, který by měl tyto hrozby odstranit [29].
Mezi možnosti implementace cloudu patří:

- Veřejný cloud
- Soukromý cloud
- Komunitní cloud
- Hybridní cloud [30]

3.3.1 Veřejný cloud

Umožňuje menším zákazníkům využívat infrastruktury velkých společností. V tuto chvíli dostatečný výkon a infrastrukturu pro vývoj AI poskytují Google Cloud Platform, Amazon Web Services, Microsoft Azure, IBM Cloud, Alibaba Cloud a další. Jeho výhodou je malá počáteční investice a jednoduché využití.

3.3.2 Soukromý cloud

Funguje jako součást jedné instituce a umožňuje např. vývojářům společnosti přistupovat ke společným zdrojům firmy.

3.3.3 Komunitní cloud

Je to sdílený cloud mezi více spolupracujícími subjekty. Může být například využíván vědeckými týmy, které spolupracují na stejném problému z jiných koutů světa.

3.3.4 Hybridní cloud

Spojuje veřejný cloud se soukromými úložišti. Pro využití embedded zařízení je asi nejzajímavější, protože se zde nabízí možnost velkých firem vytvořit si vlastní sítě, které sice poběží na embedded zařízení, ale vytvářet se budou jinde. Může tak docházet k neustálému vylepšování sítě a zároveň k rychlému přístupu zařízení

k datům velkých objemů. Na telefonu může například běžet část sítě, která bude rozpoznávat objekty v obrazu a síť v cloudu může objekty identifikovat a popsat.

3.4 Procesory (CPU)

Před tím, než nastala éra grafických karet a specializovaných chipů pro hluboké učení se neuronové sítě vyvíjely na klasických počítačových procesorech a i v dnešní době má tento způsob pořád svoje místo. Znamená to, že existuje velké množství kvalitních knihoven, které mimo jiné umožňují dobrou optimalizaci a paralelizaci chodu neuronové sítě na procesoru. Krom toho procesory poskytují nejlépe dostupné řešení.

Některé způsoby optimalizace se například můžeme dozvědět v článku Improving the speed of neural networks on CPUs [34].

Intel ohlásil, že ve 3. – 4. čtvrtletí roku 2019 má vyjít na trh nový procesor Intel Nervana, určený pro trénovací a inferenční fázi DL. Intel na jejich stránkách prohlašuje, že jejich nový procesor má mít největší efektivitu výkon/spotřeba ze všech komerčně dostupných produktů.

3.5 DSP

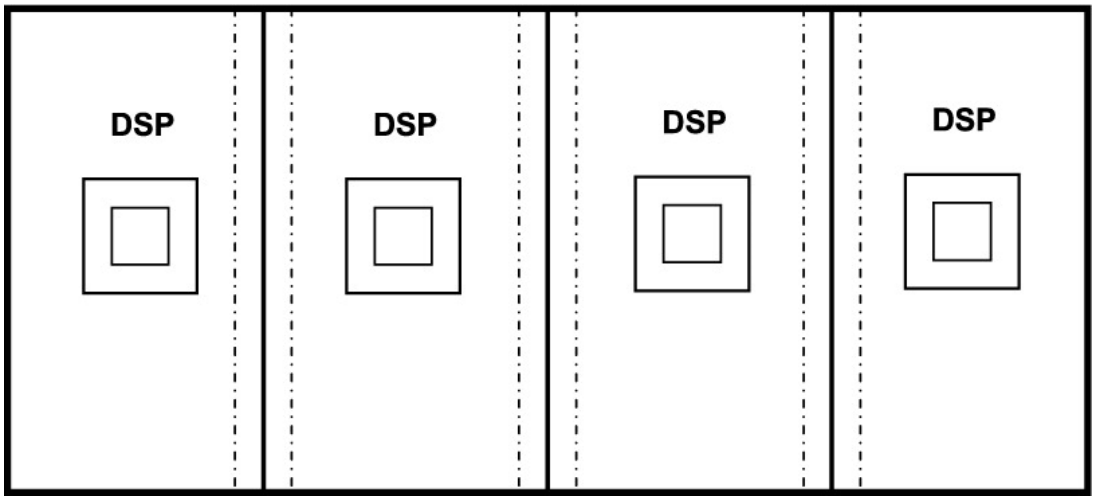
DSP, neboli Digital Signal Processor je specializovaný typ procesoru, který je vytvořen pro práci se signály. To znamená, že je schopen pracovat s hodně daty najednou a jeho instrukční sada je co nejvíc přizpůsobena práci s vektory a maticemi. Tyto procesory například umí provést operaci typu vynásob a přičti (multiply and accumulate) během jediné instrukce [31], což je využíváno například v počítání konvoluce v CNN. DSP bývá postaven na Harvardské architektuře, která umožňuje větší propustnost dat.

Signálové procesory se skvěle hodí pro počítání konvoluce u snímků s vysokým počtem pixelů. Můžeme potom každé části snímku přiřadit jeden DSP, který bude počítat konvoluci v dané oblasti velmi rychle díky přizpůsobeným instrukcím [32] (viz Obrázek č. 3.1).

DSP mohou být relativně levnějším řešením implementace jednodušších CNN, když nemůžeme využít výkonnějších řešení jako jsou GPU nebo TPU. Vhodné využití by také bylo pouze pro rozpoznávání jednoduchých struktur jako jsou hrany nebo linie ve snímcích s vysokým rozlišením nebo pro akceleraci jiných neuronových sítí [33].

Zajímavé je zejména využití v SoC (system on chip), které jsou v deep learning přímým konkurentem FPGA desek. Výhody jako je malá spotřeba energie, efektivita spotřeba/výkon nebo přímo konfigurovatelné I/O jsou zřejmé. Také v této oblasti probíhá razantní vývoj, který je popoháněn obrovskými investicemi.

To se děje především proto, že se tyto chipsety vyvíjejí pro trh s mobilními zařízeními, tzn. smartphony, kde má hluboké učení široké využití.



Obrázek 3.1: Rozdělení snímku pro využití struktury více signálových procesorů [32]

3.5.1 DSP Case study

Signálové procesory samy o sobě obecně nejsou v hlubokém učení využívány tak často, jako ostatní hardware prezentovaný v této práci. Jiný případ jsou SoC obsahující DSP.

Článek Optimizing Covolution Neural Network on the TI C6678 multi-core DSP [54] jedná o optimalizaci konvoluční neuronové sítě při implementaci na DSP. Použitý procesor zvládne 4 operace násobení s 32bit číslem s pohyblivou čárkou za jeden takt. V práci byl vyvinut vlastní Framework pro hluboké učení, protože známé frameworky jako TensorFlow, Caffe, MXNet a další používají mnoho systémových knihoven, které se nedají na DSP správně implementovat.

Výpočetně nejnáročnější operací při inferenci je konvoluce, proto se práce soustředí hlavně na její optimalizaci. Ta v testech vyšla nejrychleji při použití Winogradského algoritmu. V pozdějším porovnání s algoritmem GEMM, který například pro optimalizaci používá Caffe, vyšel čas konvoluce s Winogradským algoritmem až 17x rychlejší. Dále bylo optimalizováno řízení paralelních procesů na více jádrech díky použití semaforu místo dříve použitého přerušení.

V práci vyšlo najevo, že díky použití vlastního frameworku a optimalizacím ušitých přímo na míru danému DSP se zkrátil výpočetní čas pro inferenci 30-60x a DSP díky tomu může konkurovat FPGA z hlediska efektivity spotřeby/výkonu.

3.6 FPGA

FPGA (field programmable gate array), neboli programovatelné logické pole je deska, která obsahuje velké množství logických hradel, která se dají přeprogramovat. FPGA vykazují výborné výsledky v paralelním programování, pak také mají obecně násobně menší spotřebu energie, jsou menší, levnější a jejich architektura není pevně daná, takže oproti například procesorům i grafickým kartám může být efektivnější. Také se dají snadněji připojit různá další rozhraní, jako je I2C nebo SPI nebo další řídící obvody. [36]

Nevýhodou je zpravidla menší takt, který je ale vyvážen možností provést například maticové operace pouze v několika instrukcích, oproti CPUs a GPUs, kde jsou na to potřeba i stovky instrukcí. FPGA také nemusí tak často načítat data z externí paměti, protože se právě zpracovávaná data ukládají přímo na desce v RAM blocích.

To všechno nahrává k jejich využití pro neuronové sítě. Pro toto využití je nejvíc výhodné použít FPGA s DSP a také se speciálními RAM bloky (BRAM). DSP se využívají díky jejich rychlým operacím s maticemi a BRAM slouží jako rychlá cache paměť mezi těmito bloky. Pro větší neuronové sítě se také dá propojit více FPGA desek, kdy každá deska může zastupovat další vrstvu sítě.

Pro ušetření pamětí potřebnou k ukládání vah a také k lepší paralelizaci procesů se u FPGA používá redukce šířky slova počítače. Bylo prokázáno, že použití 16bitové aritmetiky má pouze minimální vliv na přesnost predikce oproti 32bitové aritmetice. Toto redukování vedlo dokonce až k použití binárních neurálních sítí (BNN), které využívají binárních vah s hodnotami -1 nebo +1. [37]

O porovnání FPGA a klasických procesorů pojednává práce Convolutional Neural Network on Embedded Linux System-on-Chip, která byla publikována jako technická zpráva o využití embedded zařízení pro americkou armádu [16].

Pro jistá embedded zařízení o kterých víme, že se u nich někdy v budoucnu nebude nahraná síť měnit a která slouží například k rychlé detekci objektů se jeví implementace sítě na FPGA téměř jako ideální řešení. Při zahájení masové výroby už předprogramovaných desek klesá výrazně cena, ale vývoj desky, která bude sloužit pouze pro jednu aplikaci neuronové sítě, je drahý. Tato PCB se nazývají ASIC (Application specific integrated circuit) a vykazují velký výkon s malou spotřebou energie.

Intel v řadě svých FPGA Arria 10 a Stratix 10 vyvinul inovativní způsob pro změnu šířky slova pro různé části neuronové sítě a díky tomu tyto FPGA dosahují vyšších taktů a mají výborné výsledky s malou spotřebou energie. V roce 2015 po představení těchto desek proběhlo porovnání desky Arria 10 GX1150 s GPU Titan X. Testovalo se, kolik obrazů za sekundu dokáže neuronová síť zpracovat.

Výsledkem byla podobná efektivita, ale spotřeba energie FPGA Arria 10, byla méně než poloviční [38]. Dalším významným výrobcem FPGA je firma Xilinx.

Problémem implementace NN na FPGA může být složitější programování, které je více propojené s hardwarem. Proto byl vyvinut framework OpenCL, založený na jazyku C, který interakci s hardwarem značně omezuje a umožnuje (nejen) FPGA programovat stylem vyšších programovacích jazyků.

3.6.1 FPGA Case Study

Ve většině článků o implementaci hlubokého učení na FPGA se mluví hlavně o redukci přesnosti aritmetických operací a o jejich vlivu na celkovou kvalitu inference. Také se zde oproti jiným platformám často využívají architektury jako SAE nebo SVM.

V práci Accelerating Binarized Neural Networks“ comparison of FPGA, CPU, GPU, and ASIC [57] se autoři zaměřili na implementaci binární neuronové sítě (BNN) na FPGA Aria 10 a na vlastní ASIC 14nm. Dále potom autoři provádějí porovnání výkonu a efektivity dané sítě oproti implementaci na CPU Intel Xeon, GPU NVidia Titan X a mobilní GPU Nvidia TX1.

Při implementaci velkých modelů se složitou architekturou s mnoha vrstvami, jako je například VGG, na FPGA vzniká problém s uložením vah na BRAM na desce. Tyto paměti totiž nemají vysoké kapacity a jedna vrstva s vahami uloženými v 32bit formátu s plovoucí čárkou dosahuje např. u VGG velikosti 64MB. V případě nahrazení 32bit float binárními hodnotami +1 a -1 se tato velikost sníží na pouhé 2MB a tím pádem na FPGA nemusí dojít k pomalejšímu přístupu do vnější DRAM paměti. Také tím vzniká možnost použití jednoduchého XNOR hradla místo násobičky. Na obrázku 3.2 je vidět architektura HW akcelerátoru pro tuto síť. Využití XNOR hradel místo klasické násobičky je zde dobře viditelné v části (c) Binarized multiply using xnor and popcount.

Použité FPGA Altera Aria 10 obsahuje ~6MB paměti v BRAM a 1518 DSP jednotek a takt je 150MHz. Jeho výkon ve špičce byl pro danou BNN změřen 9,8 TOP/s (Tera operace za sekundu).

Navržený ASIC byl vyroben technologií 14nm, obsahuje ~4MB paměti v BRAM a má takt ~1GHz. Jeho výkon ve špičce byl změřen 16TOP/s.

Dalším krokem bylo porovnání softwarové implementace stejné sítě na CPU a GPU. Autoři napsali instrukce pro počítání s binárními vahami a vytvořili 5 modelů odvozených od AlexNet, VGG a NT s binárními vahami.

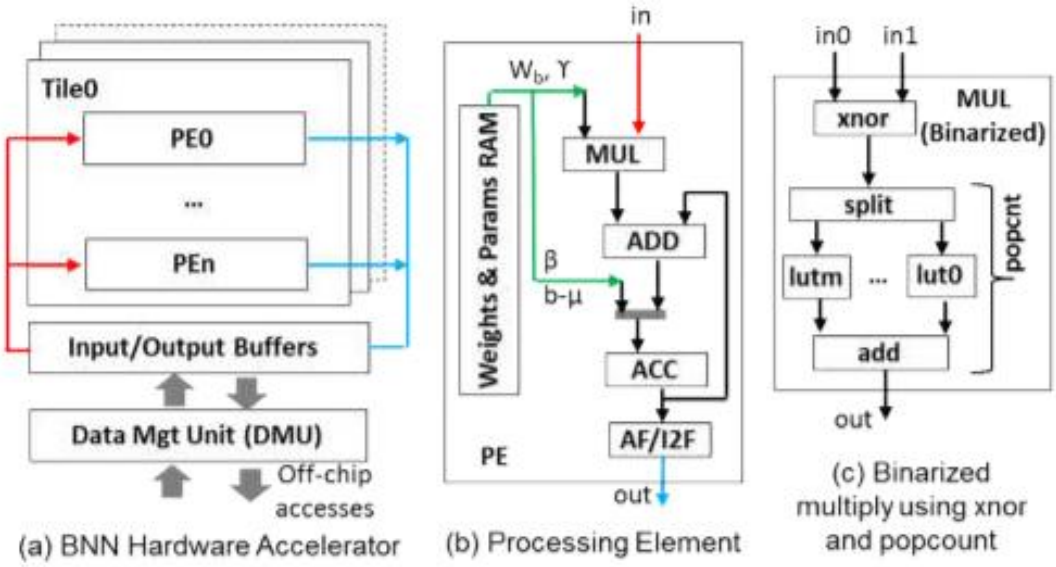
Porovnání proběhlo ve 2 kategoriích: výkon, vztažený oproti CPU; efektivita, tzn. výkon/spotřeba.

Všechny implementace sítě s binárními vahami dopadly u všech modelů ve všech parametrech lépe než implementace s 32bit float vahami. Nejvyšší výkon byl

naměřen u navrženého ASIC, kdy dosahoval u všech modelů hodnot okolo 100x větší účinnosti oproti CPU. Na druhém místě se umístila implementace binarizované sítě na GPU Titan X s průměrně 80x vyšším výkonem než CPU a na třetím FPGA Arria s 75x vyšším výkonem. Nejhůře naproti tomu dopadlo mobilní GPU s výkonem až čtvrtinovým.

V kategorii výkon/spotřeba se jako nejlepší jasně projevily ASIC a FPGA až s 1000x lepší efektivitou než CPU. Další se s velkým rozdílem umístilo mobilní GPU s efektivitou 10x lepší než CPU.

Z výsledků je tedy jasné, že dobře optimalizovaná architektura ať už FPGA nebo vhodně navrženého ASIC má majoritní podíl na celkovém výkonu i efektivitě zařízení. Pokud jde o porovnání výkonů, můžeme však pochybovat o kvalitě implementací binárních sítí na CPU nebo GPU od autorů práce. Je možné, že při využití lepší optimalizace by výsledky dopadly jinak.



Obrázek 3.2: Ukázka implementace BNN na FPGA Aria 10 [57]

3.7 TPU

TPU, neboli Tensor Processing Unit je obecně jenom více sofistikovaný ASIC chip optimalizovaný pro práci s neuronovými sítěmi. Existuje několik „druhů“ TPU pro různé druhy práce od různých výrobců. Většina z nich sdílí podobné technologické principy, ale jsou pojmenovány odlišně.

3.7.1 Google TPU

Tento chip, představený v roce 2016 používá Google ve svých aplikacích jako jsou Google Translate, Google Photos nebo Google Street View. TPU se v současné chvíli nedá koupit, ale v roce 2018 začal Google prodávat výpočetní čas na jejich TPU v jejich výpočetních centrech.

V tuto chvíli existují 2 verze TPU. TPU a TPU v2. První verze byla navržena pro rychlejší fungování neuronové sítě při nízké spotřebě energie. Druhá verze se již dá využít i pro trénování NN. První verze totiž podporovala výpočty pouze s 8bitovými int čísly, kdežto druhá verze již podporuje i operace s plovoucí čárkou. Došlo také ke zvětšení šířky paměti z 34GB/s až na 600GB/s díky upgradu z DDR3 SDRAM na HBM (High Bandwidth Memory). Třetí generace, která byla ohlášena v květnu 2018 má být až 2x výkonnější než ta předchozí.

Fungování první generace TPU společně s různými benchmarky je popsáno bez bližších detailů implementace v práci In-Datacenter Performance Analysis of a tensor Processing Unit od Googlu [39].

TPU je optimalizován čistě pro maticové operace, proto se skládá z velkého množství MMU (Matrix Multiply unit), u druhé verze je to 32 768 jednotek, které jsou vzájemně propojeny v takzvaném systollic array. Nejdřív dojde k načtení dat do jednotek MMU a potom v každé dojde k operaci násobení a sečtení a předání výsledku rovnou dalším jednotkám bez jeho uložení [40] [41]. Google prohlašuje, že jsou jejich TPU také přímo hardwarově optimalizované a provázané s jejich knihovnou TensorFlow [42].

3.7.2 VPU, APU, IPU, NPU

VPU, neboli Vision Processing Unit je chip navržený jako akcelerátor pro práci s obrazem v oblasti strojového vidění. Hlavním výrobcem VPU je v současné době Intel s jejich chipem Movidius, který je použitý v řadě procesorů Myriad nebo v Intel Neural Compute Stick.

APU, IPU, NPU a jiné jsou jiné názvy pro chipy se stejným účelem, kterým je jednotka určená pro inferenci a práci s modely neuronových sítí. Tyto jednotky jsou využívány jako samostatné komponenty některých SoC, kterým hlavní procesor předává některé požadavky týkající se obecně zpracování obrazu, rozeznávání hlasu, atp. Dále mají tyto jednotky společné, že jejich výrobci tají know how jak vlastně uvnitř fungují, takže dále nemůžeme nahlédnout do jejich architektury. Dá se ale předpokládat, že budou založeny na podobných principech, takže budou obsahovat vlastní tensorová jádra.

Hlavním výrobcem APU (AI Processing Unit) je Mediatek s jejich SoC Helio P90 a P60, které se umisťují na prvních příčkách v různých výkonnostních benchmarcích pro použití v DL [59].

Hlavním výrobcem NPU (Neural Processing Unit) je HiSilicon, který vyrábí SoC Kirin 980, který má mezi ostatním hiendovými SoC nejlepší skóre v práci s modelem využívající váhy jako hodnoty v 16bit float [59].

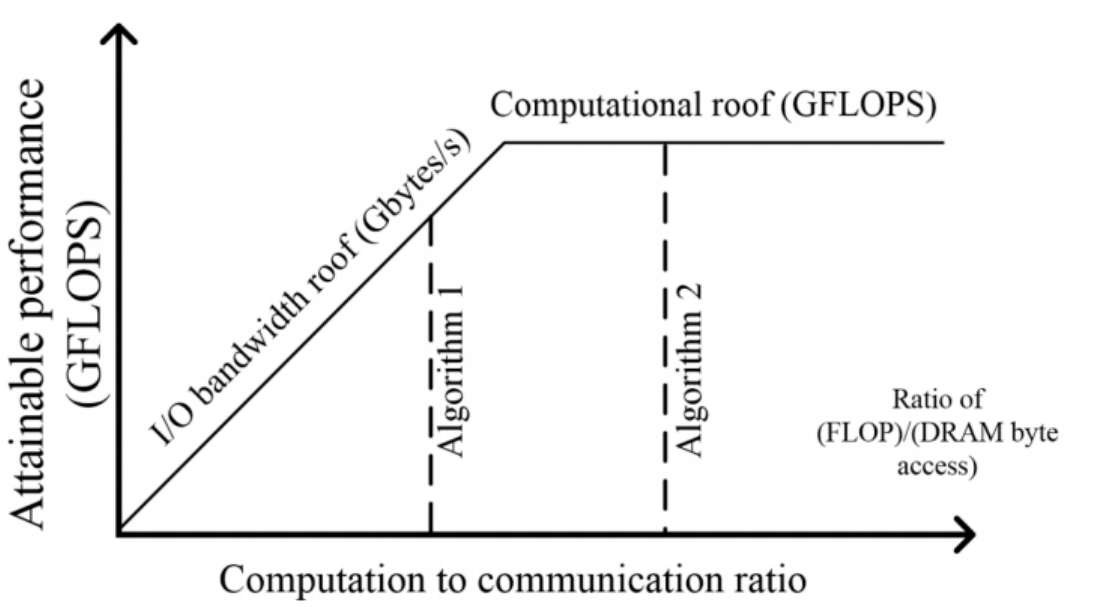
Hlavním výrobcem IPU je Google, který ho používá ve vlastním smarthponu Google Pixel 2. IPU obsahuje 8 jader, přičemž každé jádro obsahuje 512 ALU.

3.8 Grafické karty (GPU)

V dnešní době jsou grafické karty, neboli GPU asi nejvíce používaným prostředkem v DL. Přesun z CPU na GPU znamenal velký posun, protože GPU jsou určené pro pomalejší paralelizované výpočty, což se hodí pro výpočty s maticemi.

V oblasti GPU pro NN jednoznačně vede NVIDIA. Karty této značky totiž mají podporu všech frameworků pro DL, navíc obsahují podporu CUDA a také zde vznikají GPU přímo specializované pro DL například v automotive průmyslu.

V dnešní době je při použití grafických karet pro NN bottleneckem spíše než velikost grafické paměti šířka paměti (memory bandwith). Ať totiž existuje seberychlejší karta s velkou pamětí a velkým počtem jader, má jen omezenou propustnost dat, které do ní proudí pro zpracování. Na obrázku č. 3.2 je zobrazen model, vytvořený S. Williamsem, A. Watermanem a D. Pattersonem, který ukazuje na ose y dosažitelný výkon grafické karty (v GFlops) a na ose x poměr výkonu/rychlosti komunikace, tzn. šířky paměti [45].



Obrázek 3.2: Model problému šířky paměti u grafických karet [44]

Tento problém se částečně řeší využitím sítě více spojených grafických karet, kdy je každá použita pro jiný proces.

Na rozdíl od jiného hardware nacházejícího se v této práci, se grafické karty používají hlavně na trénování neuronových sítí. Pro inferenční fázi je potřeba brát v potaz i jiné přístupy. Vzhledem k tomu, že předmětem této práce je implementace aplikace využívající inferenční fázi, samotnou trénovací fázi mimo embedded zařízení se zde nezabýváme.

Výkonnými kartami vhodnými pro použití pro DL jsou v dnešní době například NVidia Tesla, Titan nebo nová herní RTX 2080Ti. Zajímavé je, že NVidia oficiálně nepodporuje použití CUDA v datacentrech na řadách karet jako jsou GTX a RTX. Pro vyhnutí se právním problémům tedy poskytovatelé musejí pořizovat karty jako je NVidia Tesla, které nemají velké výhody nad řadami RTX a GTX, ale jsou vícenásobně dražší. Tento fakt jenom ukazuje na to, že NVidia má v prostředí výrobců grafických karet pro použití DL tak zásadní postavení, že se musí zbytek trhu přizpůsobit jejich požadavkům.

V případě AMD se chystá nová Vega 20, který by měla mít vlastní Tensor Core like jádra.

3.8.1 CUDA

CUDA (Compute Unified Device Architecture) je API vytvořené společností NVIDIA, které umožňuje používat grafické karty pro obecné použití. Povoluje také přístup k virtuální instrukční sadě. Při využití se z grafické karty může s nadsázkou stát procesor, který se hodí pro práci s většími objemy dat najednou.

AMD vytváří svoje vlastní API pro použití NN ROCm Platform, které ale nepodporuje většina DL frameworků.

3.8.2 Tensor Cores

Tensor Cores je nová architektura jader GPU vyvinuta společností NVIDIA. Je vytvořena speciálně pro práci s maticemi. Podobá se MMU v TPU od Googlu.

3.9 AI Developer Kits

Sady pro vývoj AI, takzvané developer kits jsou vlastně malé specializované počítače, které jsou uzpůsobené pro vývoj aplikací využívajících hluboké učení. Využívají kombinace SoC podporujících AI, různých tensorových jednotek a akcelerátorů neuronových sítí i akcelerátorů pro zpracování obrazu, spojených

s rychlou pamětí RAM. Obvykle mají vysokou konektivitu k umožnění co nejširšího rozsahu vyvíjených aplikací.

NVidia Jetson

Příkladem může být NVidia Jetson (současná řada AGX Xavier). Toto zařízení má v sobě GPU Volta s Tensor Cores, 16GB paměti s propustností 137GB/s. Dále je vybaveno DL akcelerátorem NVDLA Engines, Vision akcelerátorem a podporuje všechny současné hlavní knihovny pro DL [46].

18. března 2019 byla oznámena nová deska NVidia Jetson Nano, která je menší, méně výkonnou a hlavně výrazně levnější (1099 vs 99 dolarů v den psaní této práce) variantou NVidia Jetson. Tato deska obsahuje prvky jako je 128-jádrové GPU NVidia Maxwell, čtyřjádrový procesor ARM A57, enkodér až pro 4k 60fps video nebo 4GB LPDDR4 paměti. Také obsahuje podporu pro chod speciálního linuxu a hlavně JetPack SDK, což je SDK pro počítačové vidění, deep learning, apod., které je klíčovou částí i na větším Jetsonu.

Intel Neural Compute Stick

Toto zařízení, které slouží jako miniaturní akcelerátor NN a které se připojuje do USB slotu, slouží pro rozpoznávací fázi (inference) pro modely neuronových sítí. Je určeno především pro aplikace počítačového vidění. Jeho hlavní součástí je totiž VPU (vision processing unit) Intel Movidius Myriad X, což je speciální procesorová jednotka navržená pro práci s počítačovým viděním. Obsahuje v sobě 16 SHAVE jader, které podporují multidimenzionální vektorové operace nebo dále také třeba výkonné obrazové enkodéry a akcelerátory, které umožňují zpracovávat až video s rozlišením 4K nebo 720p při 180 zpracovaných obrazech za sekundu.

NCS (Neural Compute Stick) podporuje používání DL knihoven TensorFlow nebo Caffe. Použití je široké. Dá se použít například v dronech, bezpečnostních kamerách, automobilových kamerách nebo pro připojení do menších embedded zařízení jako je Raspberry Pi [48].

Toto zařízení bylo například využito pro inference fázi při detekování invazivního karcinomu prsu, společně s modelem Inception V3 od Googlu, přetrénovaným datasetem IDC, obsahujícím histologické obrazy rakoviny prsu.

Coral Dev Board

Jak již bylo psáno v kapitole o 3.7, Google svoje vlastní vyvinuté TPU neposkytuje k prodeji. Vyrábí však Coral Development board, který obsahuje jako klíčovou komponentu TPU Edge, což by měla být „zjednodušená“ verze velkých Cloud TPU. Důraz je kladen na malou spotřebu a velikost a jako využití se předpokládá hlavně inference.

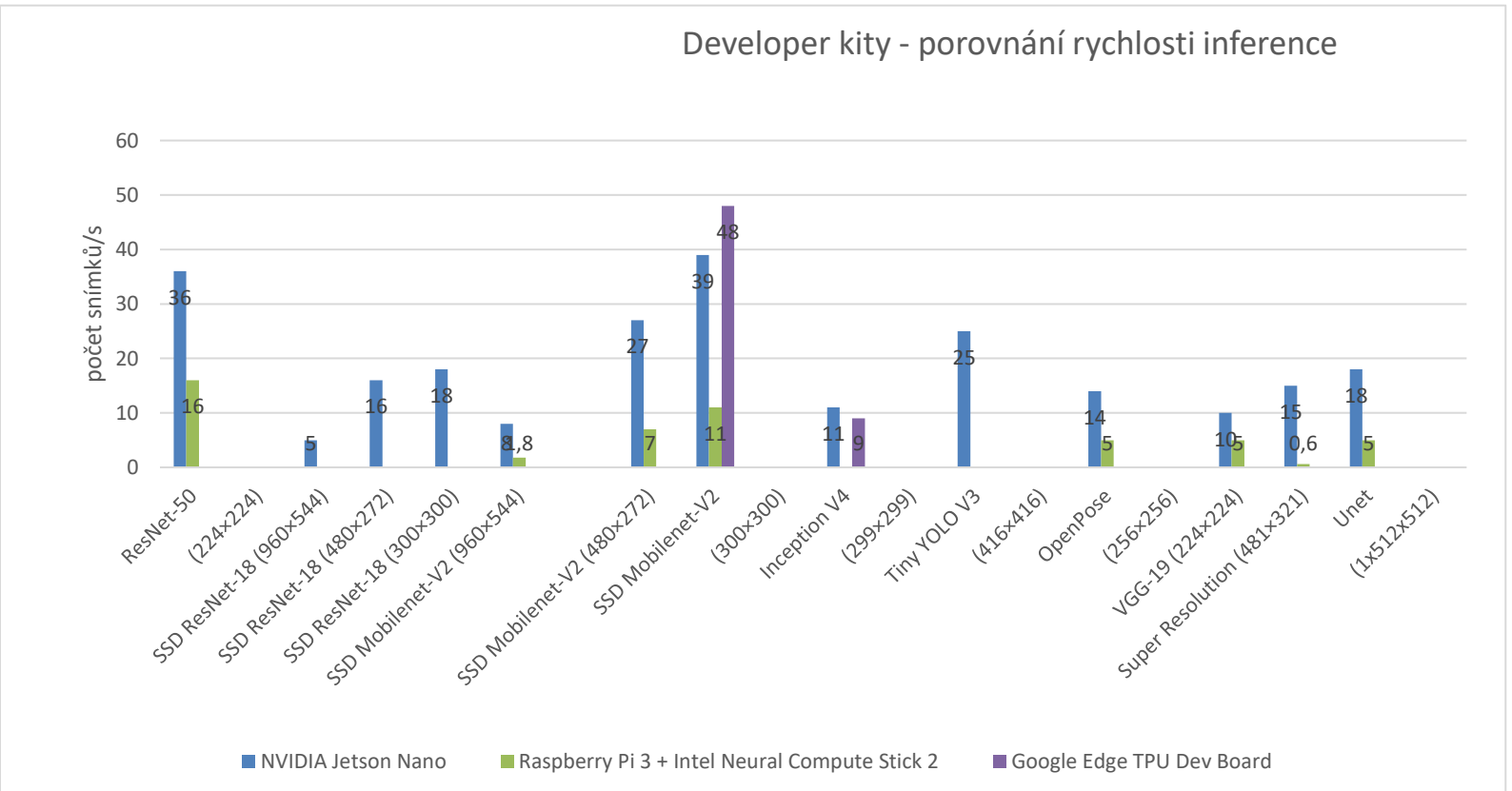
Hlavním rozdílem oproti např. Rapsberry Pi nebo jinými vývojovým deskám je absence podpory instalace Linuxu. Místo toho na desce běží vlastní operační systém Mendel vyvinutý Googlem. Ten je sice odvozený z Linuxu, ale nemá vlastní grafické rozhraní a komunikace s deskou tak probíhá pomocí SSH z jiného počítače. Hlavním podporovaným jazykem je Python.

Zajímavým prvkem je použitý procesor Freescale NXP i.MX 8M SOC. Tento procesor se totiž běžně ve vývojových deskách nepoužívá. Je to procesor pro průmyslovou výrobu, ke kterému se běžně není jednoduché dostat.

Srovnání

Na obrázku 3.3 můžeme vidět porovnání jednotlivých developer kitů z hlediska rychlosti inference v počtu rozpoznaných obrazů za vteřinu. Z obrázků je vidět, že kompletní data pro všechny zobrazené modely jsou kompletní pouze pro Jetson NANO. Jediným modelem testovaným na všech třech deskách je MobileNet V2. Nejlepší výsledek má u něj Coral dev board s Edge TPU od Googlu s 48 snímky za sekundu, potom Jetson NANO s 39 snímky a Rapsberry Pi + Intel Neural Stick 2 s 11 snímky. V tomto případě měl rozpoznávaný snímek rozlišení 300x300 pixelů.

Vzhledem k tomu, že s podobné parametry má i naše aplikace, dá se z tohoto srovnání usoudit, že nejvhodnější pro naše účely by byla deska Google Coral dev board. NVidia Jetson NANO má nicméně výkon dostatečně vysoký na to, aby posloužil stejně dobře, navíc má lepší dostupnost, podporu C++, podporu jiných DL frameworků než TensorFlow, lepší modularitu (k Coral dev board se dá například připojit pouze OEM kamera výrobce) a také cenu.



Obrázek 3.3: Porovnání rychlosti inference různých Developer kitů [58]

3.10 System-on-Chip (SoC)

System-on-Chip je kompletní sada elektronických součástek a obvodů spojených v jeden nerozdělitelný systém, který dokáže nahradit samostatný počítač. Tento systém obvykle obsahuje:
-vlastní operační systém
-jeden nebo více procesorů (CPU)
-grafickou jednotku (GPU)
-digitální signálový procesor (DSP)
-RAM a ROM paměti
-Northbridge/Southbrigde
-I/O řadiče
-AD/DA převodníky
-další moduly zajišťující např. konektivitu, power management, atd.
Velký důraz se klade hlavně na malou spotřebu a efektivitu v poměru s velikostí SoC. Hlavní využití těchto jednotek spočívá v mobilních zařízeních jako jsou smartphony a tablety. Další kategorií jsou potom minipočítače, mezi které můžeme zařadit například Rapsberry Pi.

Procesory v moderních výkonných SoC jsou postaveny na rozdíl od klasických desktop procesorů na ARM architektuře (Advanced RISC Machine). Ta pracuje s omezenou instrukční sadou s optimalizovanými rychlejšími instrukcemi.

3.10.1 SoC Case Study

Článek AI Benchmark: Running Deep Neural Networks on Android Smartphones [55] pojednává o porovnání SoC pro Smartphony s operačním systémem Android. Odvolává se na důležitost moderních AI mobilních chipsetů kvůli úkolům, které jsou úzce svázané s mobilními zařízeními, jako je rozpoznávání obrazů (image classification), tváří (face recognition), překladem do jiných jazyků nebo doplňování vět v chatovacích aplikacích. V tuto dobu je většina z těchto úkolů prováděna přes vzdálené servery, což přináší problémy s latencí sítě, ochranou soukromí nebo maximálním počtem připojených klientů k serveru.

Autoři nejdříve uvádí své předchozí pokusy implementace CNN na GPU a DSP obsažené v SoC, které byly založeny na RenderScript frameworku. V následujícím porovnání s TensorFlow Mobile knihovnou určenou pro mobilní procesory bez těchto modulů ale svoje řešení zavrhují, protože větší rychlost inference nebyla dosažena.

Dále se tedy soustředí na porovnání enginů pro hardwarovou akceleraci chodu neurálních sítí, využívajících GPU i DSP na chipu, od výrobců samotných SoC, jako je Qualcomm nebo HiSilicon (Huawei). Zvláštní důraz je kladen na Android Neural Networks API (NNAPI), které sjednocuje hardware akceleraci

neuronových sítí na SoC různých výrobců, takže se dá použít na aplikace, které nejsou závislé na použité platformě.

Qualcomm je americká společnost, která se v současnosti soustředí na SoC s názvem Snapdragon. Vyvíjí si vlastní architekturu procesorů využívajících Arm NEON instrukční sadu a také vyvíjí vlastní grafické chipy Adreno. Chipsety od Qualcommu pokrývají až 55% trhu s chytrými mobilními telefony. V roce 2015 poprvé představili Snapdragon Neural Processing Engine SDK (SNPE SDK), které podporuje hlavní DL frameworky jako jsou Caffe/Caffe2, TensorFlow, PyTorch, Chainer a a využívá při tom hlavně DSP a GPU na SoC. Qualcomm má i podporu pro NNAPI, která slouží jako API pro již vyvinuté SNPE SDK a všechny tyto nástroje volně poskytuje všem OEM zákazníkům.

HiSilicon se svými SoC Kirin nevyvíjí vlastní komponenty chipsetu. Místo toho používá běžně neprodejné procesory Arm Cortex a různé verze grafických chipů Mali. Kirin místo GPU a DSP sází na vlastní NPU (Neural Processing Unit), které ale nejsou obsažené v každé verzi chipsetu. Proto existuje podpora pro DL jen u několika nejnovějších chipů. Tato podpora sestává z HiAI Mobile Computing Platform SDK a podporuje pouze frameworky TensorFlow Mobile, Lite a Caffe. Zajímavá je podpora 16, 8 a dokonce 1 bitových modelů s plovoucí desetinnou čárkou. Modely chipsetu Kirin, které obsahují podporu DL nejsou prodejné a používají se pouze do Huawei zařízení. Od verze Androidu 8.1 Kirin nabízí NNAPI drivery, které ale omezují použití modelů na pouze 16-bitové.

MediaTek stejně jako HiSilicon používá procesory Arm Cortex a Mali GPU. Akcelerací DL na jejich chipsetech se začali zabývat až v roce 2018, kdy vypustili jejich Helio P60. Tato platforma má vlastní APU (AI Processing Unit), ale na rozdíl od Kirinu při akceleraci nespoléhá pouze na ní. Při práci s kvantizovanými modely malé velikosti se využívá právě APU, ale při práci s většími modely se používá kombinace CPU/GPU. Společně s tímto chipsetem vydal MediaTek i NeroPilot SDK, které běží s podporou TensorFlow Lite a Android NNAPI. To bohužel obsahuje jen omezenou podporu modelů s INT8 operacemi.

Samsung používá značku chisetů Oxynos u kterých používá hlavně CPU Arm Coretex a Mali GPU i když pro jejich poslední high-end chipsety vyvíjí i vlastní Mongoose CPU. Samsung má vlastní VPU (Vission Processing Unit), které používá pro AI v kamerách v Samsung smartphonech. Samsung zatím nevypustil žádné SKD ani NNAPI, které by umožňovalo využívat těchto vlastnostní jejich chipů i ostatním výrobcům.

Google po změně strategie a začátku výroby vlastního Google Pixel telefonu místo obrandovaných Google Nexusů přišel s vlastním Pixel Viseal Core AI chipem. Tento chip obsahuje Arm Cortex CPU, LPDDR4 RAM a 8 IPU (Image Processing

Unit), dohromady dosahující výkonu až 3,2 TFLOPS. Google nevypustil SDK a NNAPI drivery pro další vývojáře.

Arm Cortex CPU/Mali GPU je kombinace, kterou využívá ve svých chipsetech mnoho výrobců. Pro použití DL je to výhodné, protože instrukční sada Arm NEON používaná v pokročilých CPU od Armu může částečně nahradit funkcionalitu DSP, protože podporuje instrukce rychlého násobení s vysokou šířkou slova a to i s proměnnými s plovoucí desetinnou čárkou. Technologie DynamiIQ podporuje paralelní výpočty na jednojádrovém procesoru. Arm také představil vlastní Arm NN SDK, které poskytuje hardwarovou akceleraci ve spolupráci s GPU chipy Mali a pracuje s TensorFlow, Caffe, ONNX, a TensorFlow Lite. Tyto funkcionality jsou zároveň podporovány v Andorid NNAPI.

Pro porovnání těchto chipsetů autoři v práci vyvinuli vlastní benchmark s 8 testy:
1) Rozeznávání obrazů s modelem MobileNet V1
2) Rozeznávání obrazů s modelem Inception V3
3) Rozeznávání obličejů s modelem Inception-Resnet V1
4) Odstraňování Gaussovského šumu
5) Rekonstrukce originálního obrazu z jeho downscaled verze s modelem VGG
6) Rekonstrukce originálního obrazu z jeho downscaled verze s modelem ResNet
7) Rozeznávání obrazu sémantickou segmentací
8) Vylepšení obrazu

Všechny tyto sítě jsou navrženy tak, aby jejich implementace mohla proběhnout i na zařízení s menším výkonem. V porovnání se autoři nesoustředí na výkon jednotlivých chipsetů jako celku, ale jen na výkon v oblasti DL.

SoC	Cores	Test 1, ms	Test 2, ms	Test 3, ms	Test 4, ms	Test 5, ms	Test 6, ms	Test 7, ms	Test 8, ms
HiSilicon Kirin 970	CPU (4x2.4 GHz A73 & 4x1.8 GHz A53) + NPU	160	132	2586	274	240	4848	742	193
Mediatek Helio P60 Dev	CPU (4x A73 + 4x A53) + GPU (Mali-G72 MP3) + APU	21	439	2230	846	1419	4499	394	1562
Exynos 9810 Octa	8 (4x2.7 GHz Mongoose M3 & 4x1.8 GHz Cortex-A55)	149	1247	1580	956	1661	2450	613	1230
Snapdragon 845	8 (4x2.8GHz Kryo 385 Gold & 4x1.8GHz Kryo 385 Silver)	65	661	1547	1384	3108	3744	362	1756
Exynos 8895 Octa	8 (4x2.3 GHz Mongoose M2 & 4x1.7 GHz Cortex-A53)	135	742	1548	1213	2576	3181	451	1492
Snapdragon 835	8 (4x2.45 GHz Kryo 280 & 4x1.9 GHz Kryo 280)	97	855	2027	1648	3771	4375	439	2046
Snapdragon 820	4 (2x2.15 GHz Kryo & 2x1.6 GHz Kryo)	119	839	2074	1804	4015	5055	410	2128
Nvidia Tegra X1	4 (4x1.9 GHz Maxwell)	102	925	2328	1811	3824	4437	384	2161
Snapdragon 660	8 (4x2.2 GHz Kryo 260 & 4x1.8 GHz Kryo 260)	115	1025	2299	1806	4072	4695	547	2225
Snapdragon 636	8 (8x1.8 GHz Kryo 260)	110	1055	2405	1910	4271	4877	515	2330
Exynos 8890 Octa	8 (4x2.3 GHz Mongoose & 4x1.6 GHz Cortex-A53)	139	1810	3314	1536	3594	4717	937	2148
HiSilicon Kirin 955	8 (4x2.5 GHz Cortex-A72 & 4x1.8 GHz Cortex A53)	136	1383	2932	2143	5132	6202	751	2731

Obrázek 3.4: Porovnání SoC [55]

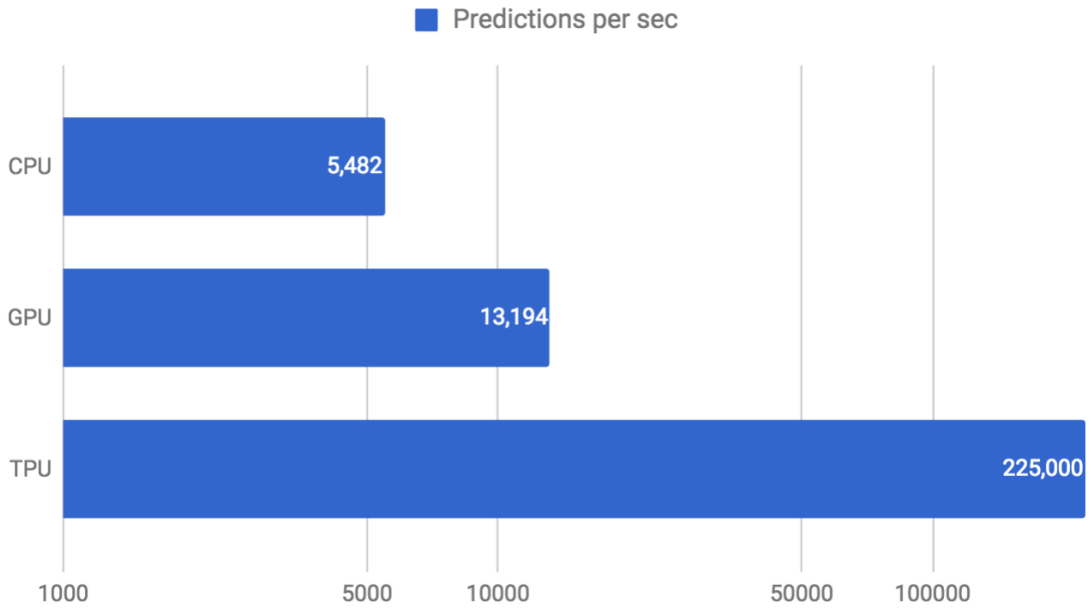
V obrázku 3.3 vidíme výsledek porovnání nejvyšších řad SoC různých výrobců ve výše definovaném benchmarku. Z něj vidíme, že většina SoC má svoje silné stránky

v jiných disciplínách (zmiňovaných testech). Například HiSilicon Kirin 970 vyniká v rozeznávání obrazů s větším a přesnějším modelem InceptionV3, ale s menším modelem MobileNetV1 za ostatními high-end SoC spíše zaostává.

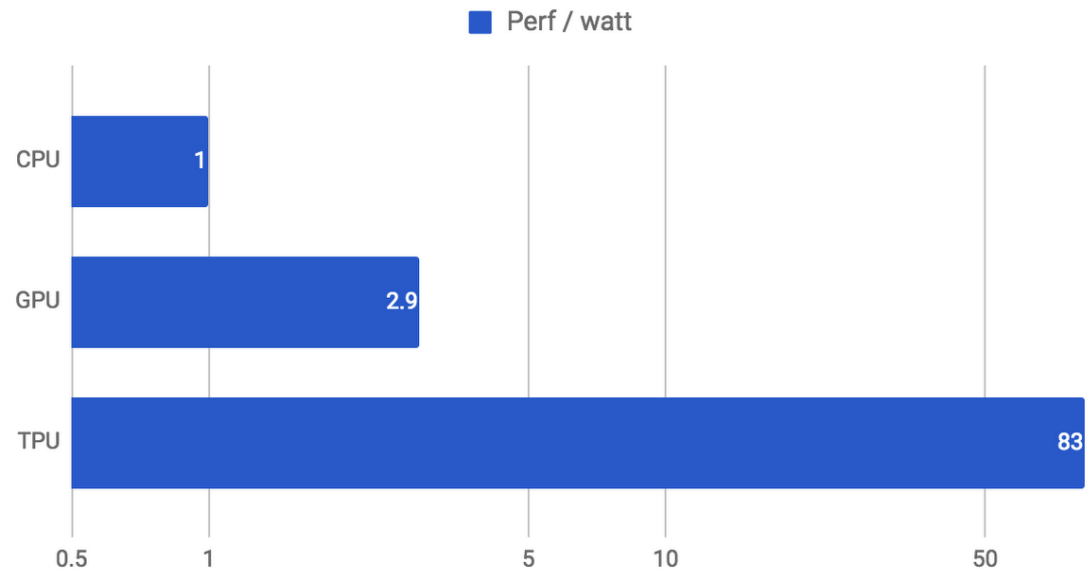
Vítězem tohoto testu se stal SoC HiSilicon Kirin 970, který vynikal především v testech 4, 5 a 8, kde ostatní SoC násobně překonával. Tento vysoký výkon je způsoben použitými NPU. Výkon by se mohl ještě zvednout po implementování i jiných než 16bit modelů do Android NNAPI.

Druhý v řadě, Snapdragon 845 od Qualcommu, se oproti HiSiliconu opírá hlavně o výkon svého CPU. Oblastmi, kde vyhrává nad vítězem jsou ale hlavně testy, kde se dá využít kvantizace modelu, což NNAPI od HiSiliconu nepodporuje. Jmenovitě je to tedy hlavně test 1, kdy je má Snapdragon 845 hodnotu rychlosti inference 21ms oproti 160ms Kirinu 970.

3.11 Srovnání



Obrázek 3.5: Porovnání CPU, GPU, TPU: Predikce za vteřinu [43]



Obrázek 3.6: Porovnání CPU, GPU, TPU: výkon/spotřeba [43]

Na obrázku 3.4 a 3.5 vidíme hrubé porovnání třech základních zkoumaných platform, respektive třech základních hardwarových architektur. Kdybychom do těchto obrázků měli doplnit FPGA, spadalo by zřejmě někam mezi GPU a TPU, ale takové srovnání nelze obecně provézt, protože u FPGA je výkon velmi závislý na konkrétní implementaci. SoC je kombinací všech přístupů, a proto také nelze do tohoto obecného porovnání zařadit. DSP spadá pod kategorii CPU, i když se jedná o speciální případ. Obecně ale platí, že se DSP v DL samostatně nepoužívá a slouží jenom jako podpůrná komponenta systémů sestávajícího z většího počtu součástí.

Zcela samostatnou kategorií jsou potom Developer kity, které slouží jako hodnotný nástroj výrobců pro předvedení technologií a pro vývoj. Tyto kity je obtižné porovnávat s ostatními přístupy, protože každý může být navržen jinou metodou a obsahovat jiné komponenty. Tyto kity by se tedy měly porovnávat pouze s ostatními takovými kity. Toto porovnání je obsaženo v části 3.9 Developer Kits.

Z porovnání vychází nejlépe TPU s nejlepší efektivitou výkon/spotřeba a i celkovým výkonem. Tento výsledek je očekávatelný, s ohledem na to, že TPU je z porovnávaných přístupů přístupem nejmodernějším a také nejúžeji zaměřeným pouze na DL.

Celkové porovnání předchozího hardware můžeme rozdělit do několika oblastí. Především se jedná o latenci, spotřebu energie/efektivitu, flexibilitu, efektivitu paralelních procesů (parallel Computing) a výkon. Také do tohoto srovnání nebudeme zahrnovat TPU od různých výrobců, ale zjednodušeně je můžeme zařadit ve srovnání pod ASIC.

3.11.1 Latence

V tomto směru dominují FPGA a ASIC a to hlavně z důvodu, že oproti GPU, CPU nebo SoC na nich neběží samostatný operační systém a samotné instrukce probíhají tedy rovnou bez jakékoliv prodlevy nebo přerušení. Vyplívá tedy, že pro aplikace, které vyžadují inferenci v reálném čase (tzn. zpracování videa, detekci objektů za jízdy, atd.) se hodí implementace na FPGA nebo ASIC.

3.11.2 Spotřeba

Jednoznačně nejlepší spotřebu a zároveň efektivitu spotřeba/výkon vykazují znovu FPGA a ASIC. Je to způsobeno přístupem implementace, kdy je samotná neuronová síť blíže HW a je tedy lépe optimalizovaná než při využití high level API. Na druhém místě je potom SoC, který se v některých případech blíží efektivitě FPGA nebo ASIC.

Tento parametr je pro implementaci na embedded zařízení klíčový, ale vhodnou optimalizací se dá ještě dále snižovat.

3.11.3 Flexibilita

Pod tímto termínem si můžeme představit obtížnost implementace různých architektur nebo změnu modelu v průběhu životního cyklu zařízení.

V tomto ohledu nejlépe vychází GPU a CPU z hlediska obtížnosti programování díky množství kvalitních, optimalizovaných a spolehlivých API. Zejména NVidia má výbornou podporu pro svoje GPU. Podpora pro SoC v podobě vydávanách SDK od jejich výrobců ovšem do budoucna bude znamenat stejně kvalitní API i pro SoC a to zvláště v případě, jestli že dojde ke sjednocení těchto API v podobě jednotného Android NNAPI.

FPGA jsou při požadavku na co nejlepší optimalizaci těžko programovatelné, protože se při tom musíme spoléhat na jazyk VHDL. (I když v poslední době se výrobci snaží vyvinout API, které by umožňovalo toto programování ve vyšších jazycích. Např. Open-VINO od Intelu.) Oproti ASIC se to ovšem dá zvládnout v rozsahu například týdnů oproti měsícům až rokům. FPGA jsou také velmi flexibilní co se týče připojených dalších rozhraní a dalšího zpracování signálu.

Flexibilita je důležitý parametr pro dlouhý životní cyklus zařízení, protože nové architektury a modely jsou vyvíjeny velmi rychle.

3.11.4 Parallel Computing

GPU, FPGA i ASIC dokážou provádět mnoho paralelních operací najednou. Rozdíl je ve způsobu práce a její efektivitě. SoC se v tomto ohledu spoléhá na GPU, DSP nebo vlastní TPU jednotky.

GPU dokáže provádět mnoho instrukcí při jednom taktu najednou, ovšem nejdřív proběhne například hromadně postupně při konvoluci násobení, potom sčítání, atp.

FPGA i ASIC používají pipelining, kdy v jedné vrstvě běží operace násobení a mezitím v další běží operace sčítání, takže všechen HW je pořád zapojen do práce.

3.11.5 Výkon

Výkon samotné implementované aplikace záleží velmi na její optimalizaci na HW a nedá se tedy logicky říct, které zařízení vychází v tomto ohledu nejlépe.

Rychlost taktu je největší na klasických CPU, kde ale chybí větší paralelizace. Nejlépe výkonnostně vychází zatím stále ještě GPU, díky velké míře paralelizace a velkému množství jader (dnes i tensorových jader). To je ale vykoupeno velkou spotřebou energie a vysokou cenou. [56]

4 TRÉNOVÁNÍ MODELU

Naše aplikace spadá do kategorie rozpoznávání objektů (object detection). Vstupem je živý přenos z kamery (nebo z uloženého videa), který je vhodně upraven, v další fázi projde modelem konvoluční neuronové sítě a výstupem je obraz s boxy, které určují polohy a typy jednotlivých objektů. Našimi objekty, které chceme správně klasifikovat jsou 2 druhy bonbonů Maoam, každý s jinými příchutěmi.

Naším řešením je použití techniky transfer learning a přetrénování některého z veřejně dostupných modelů pro tuto aplikaci. Jako vhodný model byl vybrán podle části 2.2.1.4 Transfer learning model s architekturou MobileNetV2, který měl také v části 3.9 AI Developer kits nejlepší výsledky pro použité zařízení (NVidia Jetson Nano).

4.1 Výběr datasetu

Výběr vhodného datasetu je klíčovou částí trénovací fáze při přípravě nového modelu. Bohužel kvalitní zhodnocení velkých datasetů je obtížné vzhledem k rozmanitosti jednotlivých snímků a klasifikovaných objektů.

V práci Comparison of Visual Datasets for Machine Learning [60] se o to autoři přesto pokouší a porovnávají 8 velkých známých datasetů (ILSVRC, COCO, SUN, PASCAL VOC, INRIA, Caltech, KITTI). V datasetech ILSVRC a PASCAL VOC často podle statistiky zabírají obrázky více než 10% celkového místa na snímku. To se nehodí k našemu využití, proto tyto databáze zavrhujeme. Ze zbylých možností se nejlépe hodí k našemu využití dataset COCO.

Dataset COCO (Common Objects in Context) má potřebnou velikost (více než 200 000 obrazů s popisky objektů) a zároveň je dostatečně populární na to, abychom mohli najít vhodný model. Měl by být dostatečně přesný, protože nepoužívá ke klasifikaci objektů pouze obdélníkové boxy, ale odděluje je od okolí přesněji pomocí polygonů. Obsahuje 80 kategorií objektů, které představují základní běžné objekty, které by mělo podle autorů rozpoznat i čtyřleté dítě.

Zároveň vytvoříme vlastní dataset, který použijeme pro vytrénování našeho modelu.

4.1.1 Vlastní dataset

4.1.1.1 Prvotní návrh

Náš dataset obsahuje 216 snímků s označenými kategoriemi bonbonů Maoam s označenými objekty, viz obrázek 4.1.

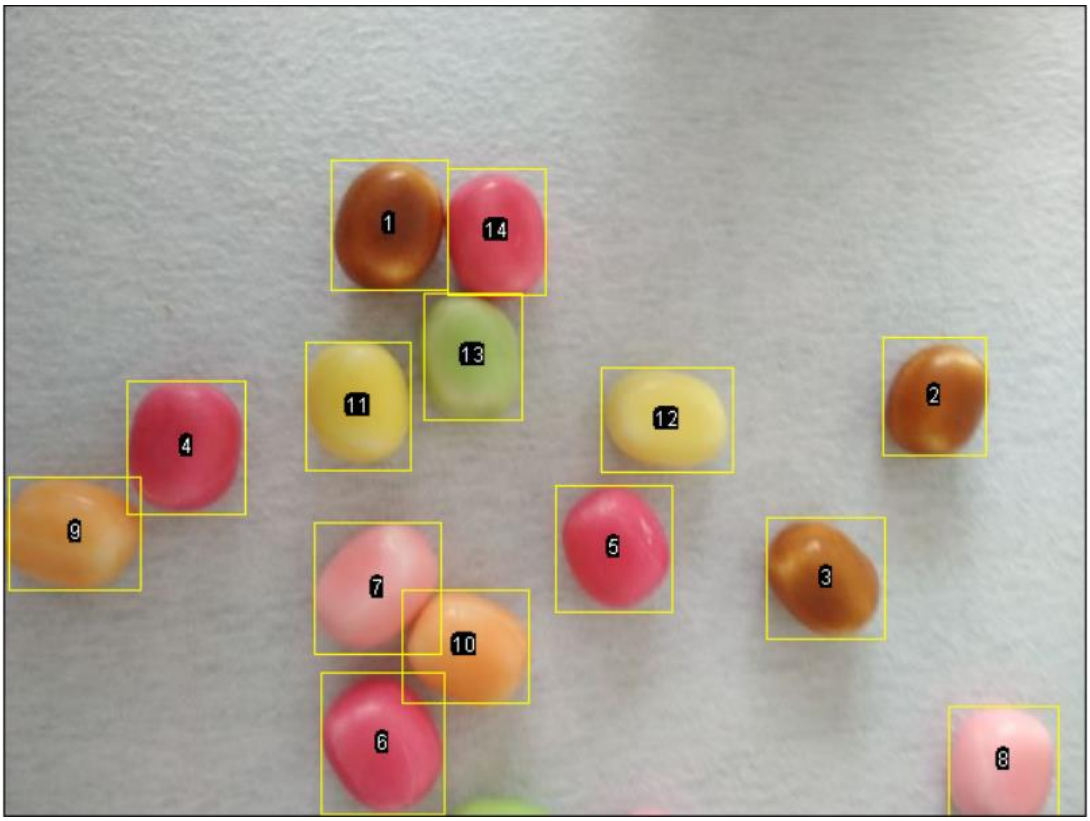
Classes	Subclasses	Mark
Crasher	green	CrGrn
	brown	CrBrn
	yellow	CrYel
	light pink	CrlPi
	dark pink	CrdPi
	orange	CrOrg
Frutis	gray	FrGry
	orange	FrOrg
	pink	FrPi
	yellow	FrYel
	ligth red	FrlRd
	dark red	FrdRd

Tabulka 4.1: Kategorie vlastního datasetu

Snímky byly pořízeny fotoaparátem Sony DSC RX100M4 a mobilním telefonem Xiaomi Redmi 4X za různých světelných podmínek. Dataset obsahuje snímky ve kterých se vyskytuje pouze objekt jedné kategorie (obrázek 4.2) i snímky, ve kterých jsou přítomny různé objekty(obrázek 4.3), které se někdy i překrývají. Dataset jsme se snažili vytvořit tak, aby byly objekty foceny na různém pozadí, z různé vzdálenosti, za různých světelných podmínek a aby na některých snímcích byly přítomny objekty podobné objektům klasifikovaných tříd. Všechny snímky byly převedeny do formátu jpeg a konvertovány do rozlišení 640x480.

Přidání popisů proběhlo ve volně šiřitelném pluginu Alp labeling tool, který je rozšířením prohlížeče snímků Fiji. Informace o objektech ve snímcích se zapisují jako textový soubor se stejným názvem jako název snímku (obrázek 4.4). Používaný formát zápisu je stejný jako pro dataset KITTI, stejně jako struktura souborů.

Obrázek 4.1: Ukázka vlastního datasetu – 1 objekt na snímek



Obrázek 4.2: Ukázka vlastního datasetu – více objektů ve snímku

```
CrBrn 0.0 0 0.0 193 91 262 168 0.0 0.0 0.0 0.0 0.0 0.0 0.0
CrBrn 0.0 0 0.0 520 196 581 266 0.0 0.0 0.0 0.0 0.0 0.0 0.0
CrBrn 0.0 0 0.0 451 303 521 375 0.0 0.0 0.0 0.0 0.0 0.0 0.0
CrdPi 0.0 0 0.0 72 222 142 301 0.0 0.0 0.0 0.0 0.0 0.0 0.0
CrdPi 0.0 0 0.0 326 284 395 359 0.0 0.0 0.0 0.0 0.0 0.0 0.0
CrdPi 0.0 0 0.0 187 395 260 479 0.0 0.0 0.0 0.0 0.0 0.0 0.0
CrlPi 0.0 0 0.0 183 306 258 384 0.0 0.0 0.0 0.0 0.0 0.0 0.0
CrlPi 0.0 0 0.0 559 415 624 480 0.0 0.0 0.0 0.0 0.0 0.0 0.0
CrOrg 0.0 0 0.0 2 279 80 346 0.0 0.0 0.0 0.0 0.0 0.0 0.0
CrOrg 0.0 0 0.0 235 346 310 413 0.0 0.0 0.0 0.0 0.0 0.0 0.0
CrYel 0.0 0 0.0 178 199 240 275 0.0 0.0 0.0 0.0 0.0 0.0 0.0
CrYel 0.0 0 0.0 353 214 431 276 0.0 0.0 0.0 0.0 0.0 0.0 0.0
CrGrn 0.0 0 0.0 248 170 306 245 0.0 0.0 0.0 0.0 0.0 0.0 0.0
CrdPi 0.0 0 0.0 262 96 320 171 0.0 0.0 0.0 0.0 0.0 0.0 0.0
```

Obrázek 4.3: Ukázka struktury značení objektů

Tento dataset byl rozdělen na trénovací část s 201 snímky a na validační část s 15 snímky. Validační část datasetu tedy tvoří cca. 7% ze všech snímků, což se prokázalo jako málo. Doporučená validační část datasetu je v odborné literatuře uváděná mezi 10-20%.

4.1.1.2 Rozšíření

Dataset byl po prvním pokusu o trénování modelu upraven a zvětšen. Také bylo nutné po přechodu na DL knihovnu TensorFlow změnit formát popisků z formátu KITTI na PascalVOC. Zároveň došlo k redukci hotového datasetu na 2 třídy bonbonů Crasher a Fruttis s přidáním třetí třídy bonbonů Bloxx.

Upravený dataset obsahuje celkově 352 snímů. Z toho je jich 298 použito pro trénovací část a 54 pro validační část. Došlo tedy ke procentuálnímu zvýšení počtu snímků z validační části na cca 15,3%. To se projevilo jako vhodný poměr.

4.2 Trénink

Trénink modelu bude probíhat s využitím metody Transfer learning. Vyhodnocení modelu probíhalo v DIGITS a v Tensorboard.

Jako ukazatel kvality modelu zde porovnáváme hodnoty mAP a loss.
mAP (mean Average Precision) je průměrná hodnota AP (Average Precission), pro jednu třídu vyjadřuje průměr Precission/Recall pro všechny obrázky, kde je objekt dané třídy detekován.

Pod pojmem loss myslíme výstup loss funkce, která vrací error při tréninku modelu.

Obecně platí, že model je tím lepší, čím je nižší hodnota loss při tréninku a čím větší hodnota mAP při evaluaci.

4.2.1 Trénink v NVidia DIGITS

Jako prostředí pro přetrénování modelu ssd_mobilenet_v2_coco byla vybrána aplikace NVidia DIGITS, která pracuje s frameworky Caffe nebo PyTorch a nabízí grafické rozhraní ke zlepšení orientace ve vývojovém prostředí a k lepšímu možnému soustředění na samotný návrh modelu. Použita byla verze DIGITS 6, ve které oproti verzím starším přibyla podpora pro trénování modelů určených pro detekci objektů nebo segmentaci objektů.

Pro přístup k většímu výpočetnímu výkonu jsme se rozhodli využít službu Amazon Web Services (AWS). Vyzkoušeli jsme při tom 2 různé AMI (Amazon Machine Instance) na 2 amerických serverech: US West (Oregon) a US East (N. Virginia).

První instance byla typu g2.2xlarge a nabízela v sobě grafickou kartu NVidia K520 s 4GB GDDR5 paměti. Zároveň v této instanci bylo předpřipraveno NVidia DIGITS, bohužel ale jenom verze 2, která nedostačovala naším účelům.

Druhá instance byla typu p2.xlarge, nabízející grafickou kartu Tesla K80. Tato karta disponuje 24GB DDR5 paměti a 4992 CUDA jádry. V této instanci byla připravena podpora DIGITS 6. Použitím AWS jsme se vyhnuli složitému konfigurování linuxu na host počítači a zároveň ještě složitějšímu instalování dependencies pro podporu Caffe, PyTorch a TensorFlow (TensorFlow využívá DIGITS v pozadí).

Nejprve je nutno v tomto prostředí vytvořit databázi ve formátu LMBD. K tomu musíme mít předpřipraven náš dataset s popisky ve vhodném formátu a s vhodným rozdělením do správných složek. Dále zde musíme definovat třídy, které dataset obsahuje a další věci, jako jsou barevná hloubka nebo enkódování obrázků.

Po vytvoření databáze jsme přistoupil k nastavení hyperparametrů modelu. Learning rate byl nastaven na hodnotu 0,00001. Tato hodnota je velmi nízká, protože jsme ji nastavili pro fine tuning, kdy náš dataset byl s porovnáním původního pouze zlomkový. Také jsme provedli pokus model vytrénovat s vyšší hodnotou parametru, ale bohužel po několika vteřinách tréninku vždy vzniknul error, který nám nedovoloval dále pokračovat. Dále jsme vybrali pro další lepší optimalizaci rychlosti učení solver Adam, který je v tomto ohledu velmi efektivní.

V další části jsme museli vytvořit vlastní (custom) architekturu sítě, protože MobileNetV2, který jsme v této části chtěli využít, není v základní sadě předtrénovaných modelů prostředí DIGITS přitomen. Jako framework, který se bude používat pro samotné učení jsme vybrali Caffe, protože má v DIGITS delší podporu (PyTorch byl přidán až později). Samotná architektura sítě se do prostředí popisuje ve formátu .prototxt. Vzhledem k tomu, že popsat celou architekturu MobileNetuV2 by bylo v daném (resp. i v jiných formátech) velmi

zdlouhavé a obtížné, převzali jsme tento popis z githubu [63] a jenom mírně poupravili počet trénovaných tříd, formát našich dat a batch size. Samotný caffemodel jsme získali stažením z [64].

100 epoch učení probíhalo 12 minut, 43 vteřin. Na tomto rychlém výsledku se silně podepsal výkon využité grafické karty. Bohužel z nám neznámého důvodu učení modelu neproběhlo podle představ. Hodnota loss velmi oscilovala v celém průběhu a po ukončení měla vysokou hodnotu (0.917357) a mAp dosahovalo téměř nulových hodnot. Po otestování několika snímků jsme zjistili, že model ani u jednoho správně neklasifikoval žádný objekt.

4.2.2 Přechod na TensorFlow

Po předchozím neúspěšném pokusu jsme se rozhodli změnit taktiku pro trénování a přejít z omezeného prostředí DIGITS k frameworku TensorFlow s nadstavbou Object Detection API, kde si můžeme všechny parametry nastavovat sami.

Výkon pro trénování modelu nám poskytovala grafická karta NVidia Geforce GTX 950M s 2GB GDDR5 paměti, 640 CUDA jádry a s CUDA compute capability 5.0. Poslední parametr určuje, jak moc je karta výkoná při použití CUDA knihovny, která je klíčovou knihovnou při tréninku modelu. CUDA uvádí, že nejnižší možná hodnota tohoto čísla pro trénink modelu je 3.0. Na to, že GTX 950M je notebooková grafika je hodnota 5.0 až překvapivě vysoká.

Jako platformu pro práci s tímto frameworkem jsme vybrali Microsoft Windows 10. Podpora je sice větší pro Linux, ale kvůli nedostupnosti žádného zařízení s tímto OS a kvůli nedůvěře k správné konfiguraci na virtuálním počítači byl zvolen právě Windows.

Pro celou práci jsme přitom využívali virtuální prostředí Anaconda, které umožňuje pracovat s oddělenou verzí Pythonu a má oddělenou systémovou struktura, do které můžeme libovolně zasahovat beze strachu o změnu důležitých dependencies nebo verzí knihoven v domácí systémové struktuře. Díky tomu jsme na OS Windows mohli pracovat stejným způsobem jakým bychom pracovali na stroji s Linuxem.

Pro práci s TensorFlow je neprve nutné stáhnout Python verze >3 (v našem případě verze 3.6.8), vývojové prostředí pro C++ do Microsoft Visual Studia (v případě instalace na Windows) a pro využití grafické karty při tréninku i NVidia CUDA správné verze podporované TensorFlow (V našem případě 10.0). Po nainstalování tohoto software můžeme teprve nainstalovat samotný framework TensorFlow (v našem případě 1.13).

Po nastavení TensorFlow bylo potřeba ještě nakonfigurovat TensorFlow Object Detection API. To pracuje s repositářem object_detection, který je součástí experimentální části /research v modelové části knihovny TensorFlow [65]. Pro

jeho správný chod je nutné stáhnout knihovny Protobuf, Cython, contextlib2, Pillow, lxml, Jupyter notebook, Matplotlib a cocoapi. Podrobnějším popisem těchto knihoven se zde zabývat nebudeme, avšak základními funkčnostmi jsou formátování dat, optimalizace, zpracování obrazu, vykreslování obrazu, práce s datasetem a webové rozhraní pro práci s Pythonem, které je zde uvedeno hlavně pro testování. Dále je nutné nastavit správnou cestu pro systémovou proměnnou PYTHONPATH, která musí ukazovat na cestu k Object Detection API.

Dalším krokem je vygenerování souborů .record z našeho datasetu. To jsme provedli využitím python skriptu poskytovaným Googlem s názvem create_pascal_tf_record.py. Museli jsme také vytvořit nový labelmap.pbtxt. Tento soubor definuje třídy, které používáme v našem datasetu a přiřazuje k nim jejich id.

V poslední fázi jsme museli nastavit samotný náš model včetně hyperparametrů a modelu z kterého bude vycházet. Vytvořili jsme celkově 2 další modely:

MobileNetV2

První trénovaný model v TensorFlow vychází stejně jako model trénovaný v DIGITS z ssd_mobilenet_v2_coco_2018_03_29 a první verze vlastního datasetu. To znamená, že je vytrénován pouze pro třídy crasher a fruttis. V našem modelu byly změněny tyto parametry:
Batch size – z hodnoty 12 na hodnotu 2, kvůli omezenému výkonu GPU GTX 950M
Solver – zvoleno RMSprop, kvůli své efektivitě při menším batch size a při práci s menším datasetem.
Počáteční learning rate – nastaveno 0.0005.

Trénink tohoto modelu probíhal 9 h 55m 55s s celkovým počtem 68 380 kroků. Velikost modelu je pouhých 18,7 MB. Z výsledků můžeme vidět, že parametr loss v průběhu tréninku velmi osciloval. To může být způsobeno použitým solverem RMSprop, který zabraňuje dlouhým plateau. Při vyhlazení hodnot ale vidíme, že celková loss se stále snižovala, a to velmi podobnou rychlostí po celou dobu. Z toho můžeme usoudit, že čas využitý k tréninku nebyl dostatečný, protože průběh loss nedosáhl při učení charakteristického tvaru exponenciální funkce. mAP dosáhlo při evaluaci hodnoty 0,9705. Bohužel při testování na testovacím datasetu jsme odhalili, že model nejspíš trpí overfittingem a jeho výsledky v reálné aplikaci tak kvalitní nejsou. To můžeme vidět na obrázku 4.4.

Obrázek 4.4: Test modelu MobileNetV2

InceptionV2

V důsledku neuspokojivých výsledků, kterých při testování dosahoval předchozí model, jsme se rozhodli vytrénovat další model, tentokrát postavený na jiné architektuře s jiným algoritmem detekce objektů.

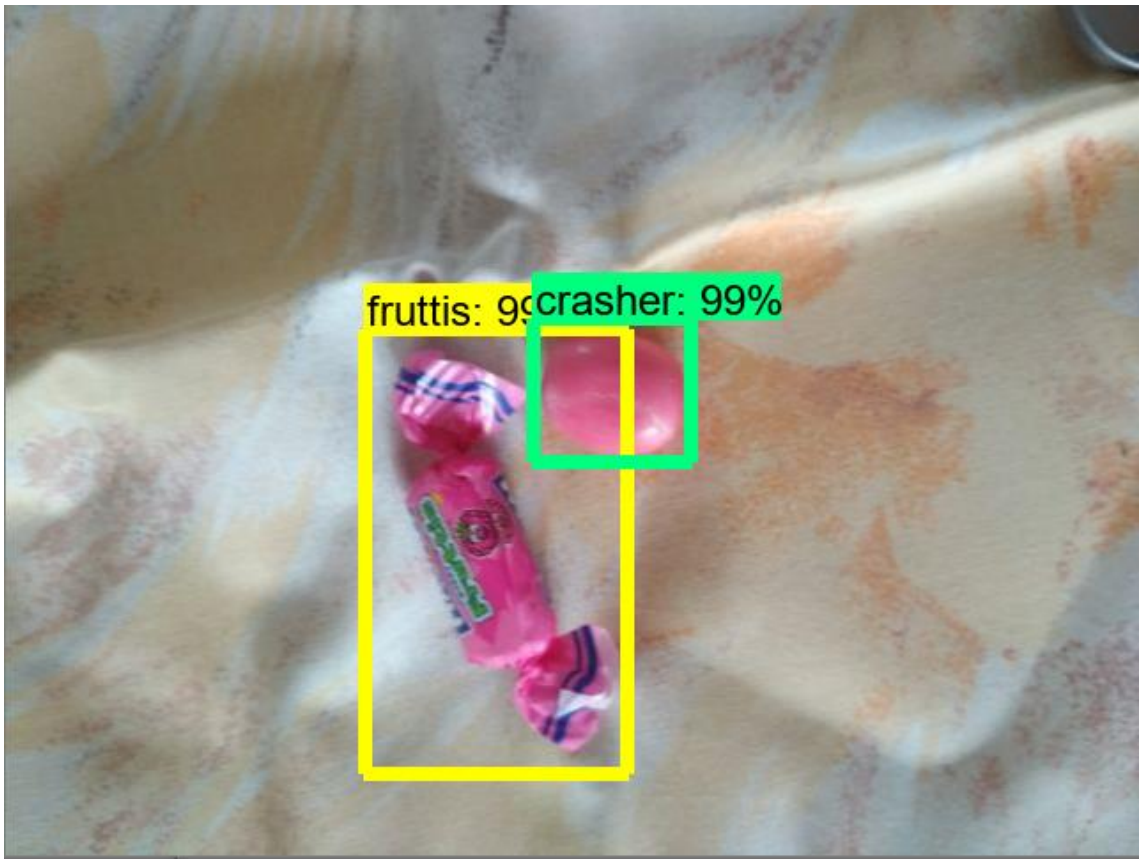
Jako architektura byla zvolena InceptionV2 vyvinutá Googlem. Model s touto architekturou by sice měl být o něco pomalejší než předchozí, ale očekávaná výpočetní náročnost pro inferenci je pořád v možnostech NVidia Jetson Nano.

Jiný detekční algoritmus jsme zvolili vzhledem k tomu, že některé naše třídy objektů (zejména třída crasher) mají pouze jednoduché příznaky jako jsou hrany, které se detekují v prvních vrstvách CNN. CDD funguje na principu detekce ohraničujících boxů v konvolučních vrstvách v celé CNN. Pokud by byly příznaky detekovány už v prvních vrstvách a dalších ne, potom by došlo k narušení algoritmu pro detekci objektů. Jako vhodný kandidát byl tedy vybrán Faster R-CNN, který by mohl pomalejší model kompenzovat aspoň rychlejší detekcí objektů (je rychlejší než SDD).

Model vychází z faster_rcnn_inception_v2_coco_2018_01_28 a z upraveného datasetu, který již v tuto chvíli obsahuje 3 třídy a rozšířenou sadu snímků.

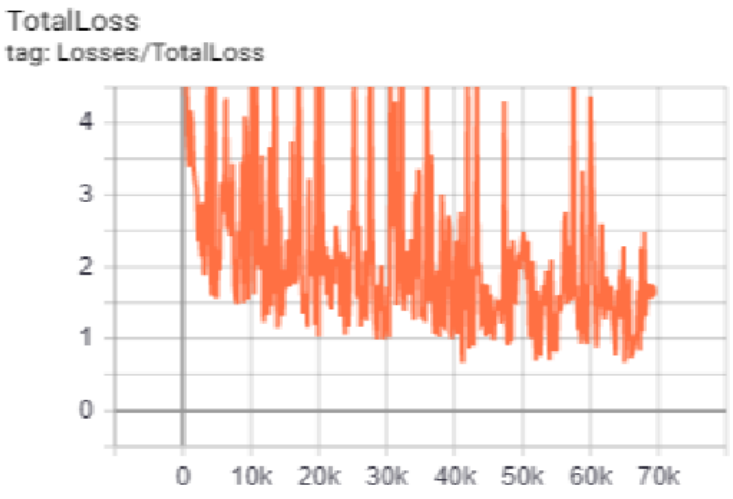
Pro trénink byly nastaveny tyto hyperparametry:
Batch size = 1
Solver – gradient descent with momentum byl zvolen, protože by měl oproti RMSprop konvergovat stále blíže k dokonalému plateau. Hodnota parametru beta (momentum) byla nastavena 0.9.
Počáteční learning rate = 0,0002; zmenšen kvůli počáteční loss, která by měla být hned od začátku teoreticky menší než u předchozího modelu.

Trénink tohoto modelu probíhal 7h 39m 56s s celkovým počtem 47 506 kroků. Konečná loss modelu dosahovala 0,01788, takže oproti minulému modelu byla řádově menší a také její vyhlazená křivka měla tvar více podobný exponenciále. Velmi vysoké bylo mAP, které se vyšplhalo na hodnotu 0,9921. Velikost modelu je 51 MB. Na obrázku 4.5 je vyobrazena detekce objektů ve snímku, který je součástí testovacího datasetu.

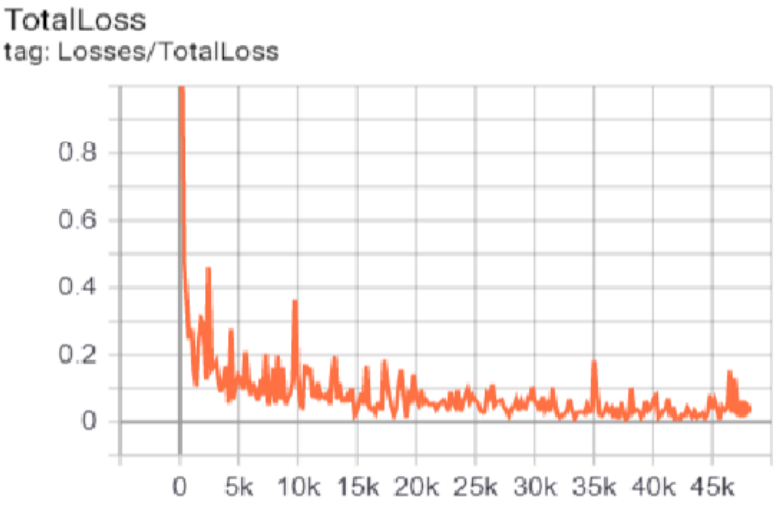


Obrázek 4.5: Test Modelu InceptionV2

MobileNetV2



InceptionV2



Obrázek 4.6: Porovnání průběhů loss funkce

Z předchozího porovnání modelů a také podle testování na testovacím datasetu vychází najevo, že model InceptionV2 je dostatečně kvalitní k tomu, abychom ho mohli použít v další části práce. Na obrázku 4.6 vidíme, jak rozdílný je průběh hodnot loss funkce v průběhu tréninku u modelu MobileNetV2 a InceptionV2.

5 IMPLEMENTACE NA ZAŘÍZENÍ

Jako zařízení, které bylo zvoleno jako nejvhodnější pro implementaci našeho modelu, bylo zvoleno NVidia Jetson Nano (viz 3.9). Hlavním důvodem pro výběr tohoto zařízení byl fakt, že to je to novinka na trhu s embedded vývojovými kity pro DL a také jeho výkonost v poměru k ceně, která byla pouze 99$. Zařízení jsme museli objednávat z americké distribuce, protože v Česku zatím není dostupné. To bohužel znamenalo dlouhou čekací dobu než zařízení přišlo a než proběhlo proclení na celním úřadu. Abychom si toto čekání zkrátili a zároveň abychom vytvořili jakýsi nástroj pro porovnání vhodného výběru zařízení, provedli jsme mezitím implementaci na zařízení Raspberry Pi model 3 B+.

5.1 Jetson Nano

Naše verze Jetsonu, s kterým jsme pracovali, není oficiální production verzí. Ta má vycházet až ve 3. čtvrtletí 2019. Místo toho se nám dostala do rukou Developer Kit verze, která v tuto chvíli slouží v podstatě jako součást beta testování a tím pádem v ní některé součásti nejsou zcela odladěné a mohou se zde vyskytovat některé chyby. O tom svědčí i široká podpora přímo od vývojářů (ne od tech supportu) na NVidia developer foru, kde vývojáři přímo nabádají uživatele k nahlašování bugů a poskytují uživatelům podporu v případě nefunkčních součástí.

Vývojové prostředí

Oživit Jetson Nano je velmi jednoduché, protože výrobce poskytuje obraz systému, který se přes program balena Etcher dá jednoduše přehrát na SD kartu, z které se potom následně systém spouští. Výhodou je, že tento předpřipravený obraz systému již obsahuje některé knihovny nezbytné pro správný chod DL aplikací. Těmito knihovnami jsou CUDA, cuDNN, TensorRT, OpenCV (mírně omezená verze, která postrádá některé funkcionality plné verze), GStreamer a další. Obraz také obsahuje SDK JetPack 4.2. Operačním systémem je L4T (Linux4Tegra), což je upravený operační systém vytvořený NVidií, který stojí na jádru Ubuntu 18.04.

Pro specifičtější build systému je možné použít NVidia SDK Manager, který jsme v našem případě nepoužili.

K další činnosti jsme museli na Jetson nakonfigurovat TensorFlow (verze 1.13) a také TensorFlow Object Detection API (viz 4.2.2).

Optimalizace modelu

Pro nejlepší výkon modelu při inferenci na této platformě je vhodné použít některý z optimalizačních prostředků jako je TensorRT nebo OpenVINO. Vzhledem k tomu, že TensorRT je vyvíjen přímo NVidií, je jasné, že na Jetsonu má větší podporu a má

více zdokumentovaných use cases. Rozhodli jsme se tedy pomocí TensorRT optimalizovat tensorflow frozen graph, který máme připraven pro inferenci (viz část 4.2.2).

TensorRT upravuje model neuronové sítě tak, aby byl přesně paralelizovaný pro GPU, na kterém později poběží inference a aby instrukce použité při běhu odpovídaly těm, které jsou na daném GPU a CUDA pro danou práci nejvhodnější. TensorRT vytvoří inference engine, který se dá později uložit ve stejném formátu jako frozen graph a stejně tak se dá i využít.

Při optimalizaci modelu TensorRT například slučuje konvoluční a ReLU vrstvy do jedné, protože GPU od NVidia obsahují instrukce, které tyto 2 procesy dokážou vykonat v jednom. Dále se například odstraňují nebo zmenšují vrstvy s nevyužitými výstupy, dochází k vytvoření strategie pro znovuvyužití paměti tensory nebo se více paralelizuje práce se vstupními daty. Hlavním zrychlujícím faktorem je ale možný převod přesnosti vah modelu. Můžeme tak model optimalizovat pro váhy FP32, FP16 a INT8 (FP = Floating Point). Využíváme tak principu blíže popsaného v části 3.6 FPGA.

V našem případě jsme si vybrali optimalizaci z přesnosti vah FP32 na FP16. Tato změna by neměla způsobit zásadní snížení přesnosti, ale zato by měla snížit výpočetní náročnost pro inferenci až o třetinu. Pro vytvoření TensorRT modelu jsme napsali skript Create_TensorRT_model.py. Způsobů pro převedení tensorflow modelů na TensorRT modely je více, ale bohužel po představení nového TensorRT API, které jsme v našem prostředí využívali ještě nevznikla nová dokumentace, kterou bychom mohli využít. Většina metod je ale upravena tak, že staré metody, funkční pro staré API již nejdou využít.

Aplikace pro detekci bonbonů

Pro detekci objektů jsme napsali v pythonu skript pro detekci objektů v real-time přenosu z kamery (Jetson_camera_object_detection.py) a další skript pro detekci objektů v uloženém videozáznamu (Jetson_video_object_detection.py). Jako kamera byla použita Raspberry Pi Camera Module V2.

Základní výhodou v práci s Jetsonem je plynulé rozdělování výpočetní zátěže procesoru nebo grafice díky sdílené paměti (CUDA Unified memory), ke které mohou obě tyto komponenty přistupovat a alokovat. HiAPI jako je TensorFlow Object detection API v Pythonu toho využívá, takže v samotném programu se tímto nemusíme dále zabývat.

Pro zachycení videostreamu z kamery jsme použili knihovnu gstreamer, která je integrovaná do L4T. Abychom jí mohli vhodně použít v pythonu, museli jsme vytvořit pipeline, která zpracuje výstup z nvarguscamerasrc a předá ho openCV, která s ním pak dále pracuje jako se vstupem z regulérního videa.

Minimální možné rozlišení, které gstreamer zpracovává je 1280x720 při rychlosti 120 FPS. My jsme proto pro naše potřeby rozlišení ještě snížili na 640x480 při 20 FPS.

Po prvním spuštění programu bylo jasné, že Jetson pracuje oproti dřívějším předpokladům příliš pomalu. Pomocí programu Tegrastats jsme následně zjistili, že je to způsobeno GPU, které mělo při běhu programu spotřebou pouze okolo 20 mW a utilization 0%. CPU oproti tomu běželo na plný výkon, kdy se jeho zátěž blížila ke 100%. Tento problém se vyskytoval pouze když jsme spouštěli inferenci z Python API. Když jsme nainstalovali příklad na detekci objektů, který NVidia poskytuje na svém GitHubu, zjistili jsme, že GPU funguje korektně. NVidia ale používá ve svém příkladu síť DetectNet, která je ale napsaná v C++ a používá místo modelu ve formátu tensorflow model ve formátu caffemodel. Po kontaktování podpory a jejich následném ujištění, že by Python API pro detekci objektů mělo fungovat jsme se snažili kolem 30 hodin čistého času zjistit příčinu problému. Bohužel jsme nic nezjistili ani po kompletním přehrání systému a rebuildu klíčových komponent. Po dalších konzultacích s podporou nám nedokázal nikdo více poradit.

Vše se změnilo po automatickém upgradu, který proběhl dne 14. 5. 2019. Po aktualizacích jsme znovu spustili program a tentokrát už GPU fungovalo správně.

Museli jsme ale řešit další problém, kterým byla systémová chyba, která se náhodně objevovala zhruba v 80% případů při spuštění programu a projevovala se totálním zamrznutím openCV okna.

Postupně jsme zjistili, že tato chyba se vyskytuje proto, že proces nvargus-daemon při vysokém vytížení procesoru spamuje systémový log hláškami o tom, že se nachází v nesprávném stavu a že nestíhá zpracovávat vstup, který přichází z kamery. Zátěž procesoru je při vyčítání z kamery vysoká, jestliže openCV stále pracuje na zobrazení posledního klasifikovaného snímku. Smyčka událostí na kompletní vykreslení nečeká a snaží se zároveň vyčíst i nový snímek z kamery. V tom případě potom dojde k pádu aplikace a systémové chybě.

Tento problém má několik možných řešení: zvětšení výkonu systému, paralelizování procesu, který řídí vykreslování obrazu a odsunutí do nové smyčky událostí nebo umělé zpomalení smyčky při vykreslování.

Zvýšit výkon systému dává i kvůli celkovému zrychlení největší smysl. Náš systém totiž neběží v nejrychlejším modu, v kernelu jsou povolena v procesoru pouze 2 jádra ze 4 a grafika má takt snížen na 852 MHz místo plných 921 MHz. Bohužel mód změnit nemůžeme kvůli napájení. Výrobce uvádí, že Jetson má v plném zatížení odběr 10W. Pořídili jsme si tedy podle dostupné dokumentace adaptér s výstupním napětím 5V a maximálním proudem 2,4A. Bohužel při použití tohoto adaptéru se ukázalo, že reálná spotřeba je vyšší a pokud odběr přesáhne

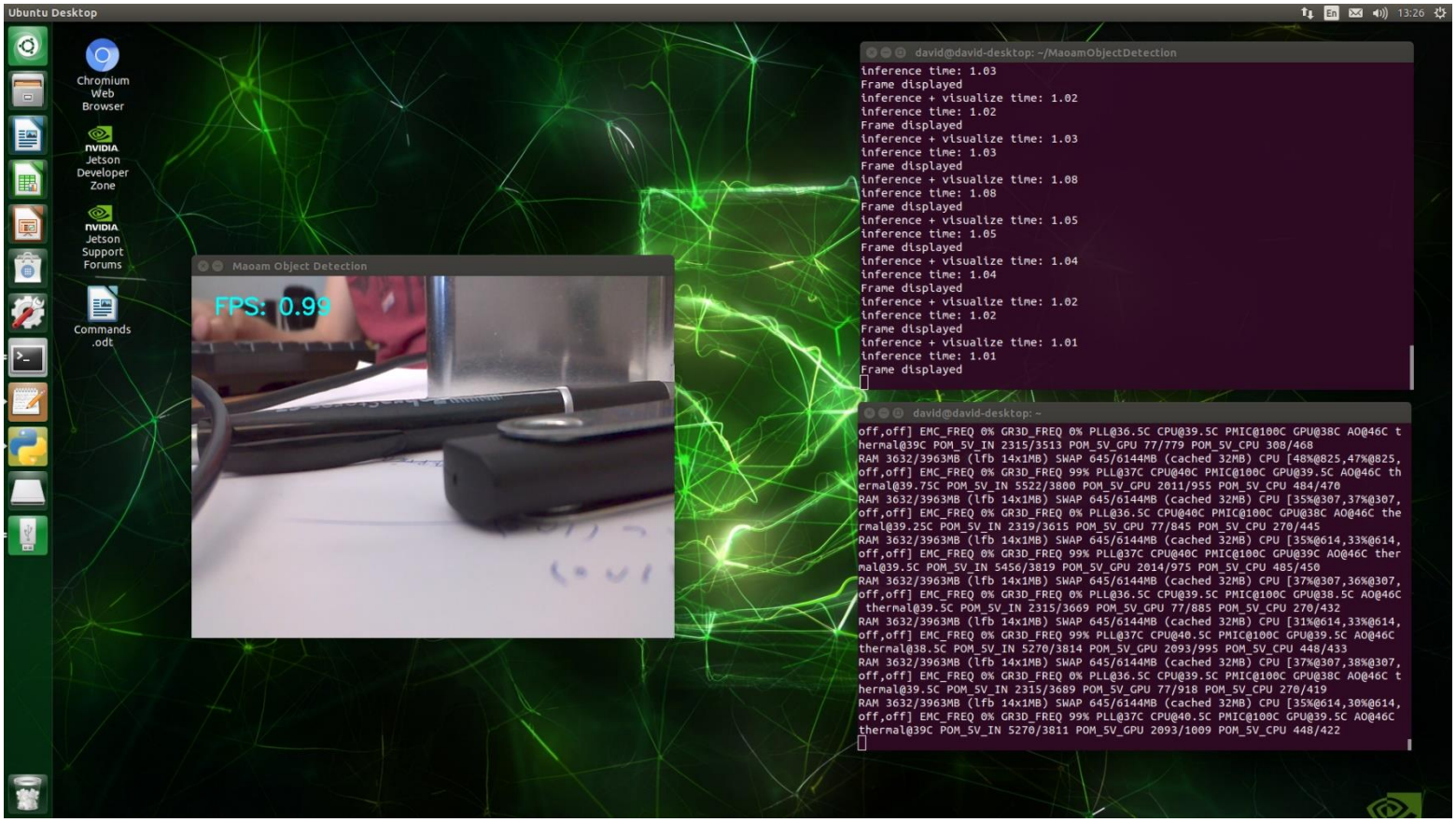
2,4A, systém se rebootuje. Musíme tedy nechat mod s maximálním odběrem 5W, při kterém zařízení může běžet cca na 75% svého výkonu. Měření vstupního proudu proběhlo multimetrem a také pomocí Tegrastats. Také jsme objednali nový adaptér s výstupním napětím 5V a proudem 4A, ale bohužel jeho vyzkoušení proběhlo až těsně před uzávěrkou této práce, takže jsme stihli vyzkoušet jenom rychlost modelu InceptionV2.

Systémovou chybu jsme nakonec vyřešili zpomalením smyčky událostí, avšak toto řešení není nijak vhodné. Dalším logickým krokem je tedy vytvoření dalšího vlákna zpracujícího obraz. Dočasným řešením je také možnost zvýšení priority procesu nvargus-daemon, ale to musí proběhnout až po spuštění procesu s unikátním ID, je to tedy velmi nepraktický způsob pro dlouhodobější používání.

Dále jsme museli zvýšit operační paměť zařízení, protože nestačila při načítání modelu a docházelo v důsledku k systémové chybě OOM (Out Of Memory). Po úspěšném načtení modelu už tato paměť navíc potřeba není. Upravili jsme tedy část paměti SD karty, aby byla použitelná jako swap paměť. Tento krok není vhodný, protože radikálně snižuje životnost paměťové karty kvůli velkému počtu zápisů a čtení, na které není karta koncipována. Pro účely vývoje je ale tento krok ospravedlnitelný.

Po těchto a dalších problémech se nám podařilo aplikaci spustit a mohli jsme přejít k testování. Přesnost detekce je vysoká a rychlost se pohybuje průměrně okolo 1 FPS (u použití modu 10W 1,3 FPS). Tato rychlost je pro tuto aplikaci použitelná a tím pádem byla implementace úspěšná. Kdybychom tento klasifikátor nasadili do výroby k pohyblivému dopravníku, tak by se daný dopravním mohl pohybovat maximální rychlostí až 0,64m/s. Navíc musíme brát v potaz, že při vypnutí GUI a při výstupu z algoritmu ve formě souřadnic objektů, by došlo k odstranění limitování výkonu procesorem a rychlost by se ještě výrazně zvýšila.

Horší je to s optimalizovaným TensorRT modelem, který je při načítání do paměti náročnější na velikost operační paměti a proto i v případě využití swap dochází většinou k OOM chybě. Zároveň u tohoto modelu trvá dlouhou dobu (průměrně kolem 18 minut), než se uložený model deserializuje. Tento problém mají podle vývojářských fór i ostatní uživatelé. V tomto důsledku jsme došli k závěru, že použití tohoto modelu není pro náš účel vhodné. Na obrázku 5.1 je zobrazena ukázka, jak vypadalo naše prostředí při testování aplikace na real-time rozpoznávání objektů.



Obrázek 5.1: Okno v T4L při spuštění aplikace

5.2 Raspberry Pi

Účel Raspberry Pi tkví především v jiných oblastech než je právě DL. Přesto o něm můžeme mluvit, vzhledem k podobné nízké cenové kategorii, do které patří i Jetson Nano, jako o jeho přímém konkurentovi. Raspberry v sobě obsahuje 64bit procesor na ARMové architektuře s taktem 1,4GHz, 1GB LPDDR2 SDRAM a podobné I/O jako Jetson, obsahující CSI patici pro kameru nebo HDMI. Navíc má ještě k tomu WiFi připojení.

Vývojové prostředí

Do zařízení jsme nahráli OS Rapsbian, což je operační systém pro Raspberry s jádrem Debianu. Dále jsme museli doinstalovat stejné dependencies potřebné pro TensorFlow, jako při instalaci na Windows (viz 4.2.2) a také TensorFlow Object Detection API a také stejně vytvořit systémové cesty pro Python. Jedinou změnou bylo, že jsme museli zkompilovat vlastní verzi knihovny protobuf, protože jinak není na Raspbian poskytována. Celý tento proces stahování, instalování a kompilování knihoven zabral asi 8 hodin a ještě se dále protáhnul, když jsme zjistili, že máme nekompatibilní tensorflow.

Aplikace

Samotné skripty Pi_image_object_detection.py a Pi_video_object_detection.py jsou podobné jako skripty které jsme použili na rozpoznávání objektů na Windows. Skript Pi_image_object_detection.py je určený k videostreamu z kamery připojené do CSI patice a následné rozpoznávání objektů ve snímku pořízeném v tomto streamu. Samotný videostream slouží pouze jako náhled, inference probíhá až po pořízení snímku. Oproti implementaci na Jetsonu tento skript používá pro zpracování obrazu z kamery třídy PiCamera, která je součástí nástrojů dodávaných s Raspbian OS a která je jednodušší na použití.

Skript Pi_video_object_detection.py potom slouží ke klasifikaci objektů rovnou z živého přenosu z kamery.

Rychlost inference se pohybuje průměrně okolo 0,1 FPS, což je pro praktické využití téměř nepoužitelná hodnota. Pro Rasberry je proto vhodnější použít méně přesný, ale zato rychlejší model, jako je třeba MobileNet, který má na něm rychlost inference průměrně okolo 1 FPS. Je také nutné zmínit, že při takto vysoké zátěži se Raspberry velmi rychle zahřívá a při době provozu delší než 15 minut hrozí restart zařízení. Pro další pokusy by tedy bylo vhodné pořídit na Raspberry nějaké chlazení. Také musí být pro načtení modelu použit swap aspoň v hodnotě 1024MB.

-	MobileNetV2	InceptionV2	InceptionV2 (10W)
Jetson Nano	4,5 FPS	0,95 FPS	1,3 FPS
Raspberry Pi	0,98 FPS	8,9 FPS	-

Tabulka 5.1: rychlosti inference

ZÁVĚR

Cílem této bakalářské práce bylo prozkoumat možnosti implementace algoritmů hlubokého učení na embedded platformu a následně provézt nejlepší možnou a dostupnou implementaci námi vytrénovaného modelu na vhodně navržené embedded zařízení. Tato zařízení jsme vybrali pro implementaci 2: Jetson Nano a Rapsberry Pi 3 B+.

V první kapitole jsme stručně nastínili počáteční vývoj strojového učení, které postupně přešlo v hluboké učení.

V kapitole 2 Deep Learning jsme nejprve definovali základní pojmy týkající se strojového a hlubokého učení a zároveň jsme zde popsali některé problémy, jako jsou například over/underfitting, které je potřeba znát pro úspěšné vytrénování vlastního inferenčního modelu. Provedli jsme rešerši některých state-of-the-art deep learningových metod a vysvětlili jak fungují a jak by se daly použít v případě implementace na embedded zařízení. Zaměřili jsme se hlavně na konvoluční neuronové sítě (2.2.1) a zejména pak na problematiku transfer learning (2.2.3) a popis jednotlivých modelů, které jsou dostupné a volně přístupné na internetu. Dalšími uvedenými metodami v této kapitole jsou 2.2.2 Rekurentní neuronové sítě (RNN), 2.2.4 Strukturované pravděpodobnostní modely, 2.2.5 Auto-enkodéry (SAE) a 2.2.6 Deep belif sítě (DBN).

V kapitole 3 Embedded zařízení jsme nejprve definovali embedded zařízení a jak ho vlastně budeme brát. Zároveň jsme poukázali na největší omezení této platformy. Dále v kapitole rozebíráme možnosti implementace na různý hardware a na cloud.

V části 3.4 Procesory (CPU) jsme uvedli výhody a nevýhody implementace na procesorech, zvláště jsme se potom zaměřili na signálové procesory (3.5 DSP), které jsou na implementaci NN svojí architekturou bližší.

V části 3.6 FPGA jsme zkoumali, v čem je výhodné použít desky FPGA a uvedli jejich fungování po implementaci. Také jsme provedli základní porovnání s grafickými kartami a provedli case study práce zabývající se binarizovanými sítěmi.

V části 3.7 TPU jsme se zabývali TPU, neboli tensor processing units a uvedli různé varianty, vyráběné některými výrobci.

V části 3.8 Grafické karty (GPU) jsme definovali některé problémy neuronových sítí po implementaci na GPU, prošli jsme důležité technologie týkající se nových GPU používaných v dnešní době pro implementaci DL a jmenovali jsme některé zástupce takových karet.

V části 3.9 AI Developer Kits jsme porovnali některé vývojářské sady pro DL a provedli srovnání, která by se nám hodila nejvíce.

V části 3.10 SoC jsme provedli analýzu dostupných SoC na trhu a provedli case study práce, která tyto SoC porovnává.

V části 3.11 Jsme provedli porovnání všech výše uvedených platform a na základně několika parametrů jsme následně zvolili, která se nejlépe hodí pro použití jako embedded zařízení.

V kapitole 4 Trénování modelu jsme přešli k praktické části bakalářské práce. Po vybrání vhodného modelu jsme vytvořili 1. verzi Maoam datasetu, která sestávala z celkového počtu 216 snímků a popisky dat byly vytvořeny ve formátu Kitti. Dataset byl připraven na rozeznávání 2 hlavních tříd, tvořených 2 druhy bonbonů a 6 podtříd, které představovaly jednotlivé příchutě. Tento dataset se neosvědčil, proto jsme vytvořili druhou, rozšířenou verzi s celkovým počtem 352 snímků a pouze 3 kategoriemi. Formát druhé verze datasetu byl změněn na formát PascalVOC.

Následně jsme nejdříve pomocí prostředí NVidia DIGITS a potom TensorFlow Object Detection API vytrénovali 3 různé modely. Model vytrénován v DIGITS se neosvědčil a dále není v práci používán. V TensorFlow jsme vytrénovali 2 modely, které vycházely z architektury SDDMobileNetV2 a Google faster_rcnn_inceptionV2. Po evaluaci modelů jsme provedli zhodnocení a došli k závěru, že model MaoamInceptionV2 je dostatečně přesný pro další použití v další části práce.

V kapitole 5 Implementace na zařízení jsme postupně popsali přípravu obou zařízení, jejich konfiguraci a následně i výkon námi navržené aplikace.

V případě zařízení NVidia Jetson došlo ještě navíc k optimalizaci modelu přímo na grafický chip Maxwell, který je na zařízení přítomen. Tato optimalizace se ale nakonec projevila jako nevhodná.

Celkově jsme vytvořili 6 skriptů, které umožňují spustit detekci objektů (bonbonů Maoam) na 3 platformách: Windows, Raspberry Pi a NVidia Jetson. Na embedded platformách skripty přebírají stream z externí CSI kamery, na Windows potom musíme dodat záznam nebo snímek, ve kterém chceme provádět detekci.

V celkovém srovnání vychází, že experiment se povedl, zejména díky tomu, že výkon aplikace na NVidia Jetson Nano je dostatečný k průmyslovému využití a pořizovací cena je oproti jiným průmyslovým řešením řádově nižší. Stejně nízká je i spotřeba energie, kdy při plném zatížení je maximální odběr okolo 12W. Při využití cloudových služeb musíme počítat na hodinu provozu s cenou okolo 1$. Při průměrné ceně energie 4 koruny za kWh se s použitím Jetsonu znovu dostáváme o řád níže. Zároveň jsou v práci vzhledem k důrazu na minimální cenu použité pouze volně šiřitelné nástroje. Z tohoto důvodu jsme ani nepoužili žádnou doplňující knihovnu, jako je například Qt pro tvorbu GUI.

Vzestup embedded zařízení využitelných v DL je masivní, ať už v podobě SoC nebo dalších zařízení jako je Jetson. Tato zařízení jsou teprve ve vývojové fázi, ale budoucích několika rocích má vyjít takových zařízení víc.

Celkově vzato můžeme říct, že Jetson Nano je kvalitní vývojářská platforma, která s cenou která je jenom mírně vyšší, zdárně překonává Raspberry Pi, ať už v rámci použití DL nebo v ostatním. Bohužel ale trpí některými dětskými nemocemi. Oproti Raspberry je také rozhodně výhodou přítomnost „dospělejšího“ Ubuntu 18.4 oproti Rapsbianu. Každopádně uvidíme, jestli se s oficiálním releasem něco změní a Jetson bude zase o něco dál. Vývojáři z NVidie na forech naznačují, že nová verze by mohla mít vylepšené napájení a nově až 16GB eMMC paměti.

Do budoucna máme mnoho možností, jak by se dala aplikace na zařízení zrychlit. V úvahu připadá zmenšení bitové hloubky pořizovaných snímků z kamery, případně až převedení do černobílé podoby.

Dále by bylo také vhodné vytvořit vícevláknovou aplikaci, která by oddělila zpracování obrazu a algoritmus pro detekci objektů. To by odstranilo problém s knihovnou gstreamer a také přineslo další pozitiva jako je budoucí lepší rozšiřitelnost a modularita.

Nakonec jsme v aplikaci rozhodli použít přesnější, zato více výkonnostně náročnější model. Toto rozhodnutí jsme udělali z toho důvodu, že pro průmyslové použití na lince (něco jako naše aplikace) se to hodí více a záleží na časové konstantě posuvu dopravníku. Proto je rychlost kolem 1 FPS ospravedlnitelná. Není nicméně od věci, kdybychom se v budoucnu pokusili dotrénovat model MaoamMobileNetV2 a případně i znovu rozšířit kategorie tak, aby model dokázal detekovat i příchutě bonbonů.

V konečném zhodnocení můžeme říct, že implementace na zařízení se povedla a aplikace funguje s dostatečnou rychlostí. Zároveň jsme představili funkční alternativu k používání cloudu nebo velkých a drahých grafických karet.

Literatura

[1] Obrázek: SAGAR, Sharma. What the Hell is Perceptron?. *Towardsdatascience.com* [online]. 2017 [cit. 2019-01-26]. Dostupné z: https://towardsdatascience.com/what-the-hell-is-perceptron-626217814f53

[2] LECUNN, Yann, León BOTTOU, Yoshua BENGIO a Patrick HAFFNER. *Gradient-Based Learning Applied to Document Recognition*[online]. 1998 [cit. 2019-01-26]. Dostupné z: http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf

[3] CULURCIELLO, Eugenio. *Neural Network Architectures* [online]. [cit. 2019-01-26]. Dostupné z: https://towardsdatascience.com/neural-network-architectures-156e5bad51ba

[4] Obrázek: COPELAND, Michael. *What's the Difference Between Artificial Intelligence, Machine Learning, and Deep Learning?* [online]. [cit. 2019-01-26]. Dostupné z: https://towardsdatascience.com/neural-network-architectures-156e5bad51ba

[5] KRIZHEVSKY, Alex, Ilya SUTSKEVER a Geoffrey E. HINTON. *ImageNet Classification with Deep Convolutional Neural Networks*. DOI: 10.1145/3065386.

[6] Obrázek: PACLÍK, Pavel. *What you should know about Deep Learning*. Prezentace

[7] Obrázek: *Role of Bias in Neural Networks* [online]. In: . [cit. 2019-01-26]. Dostupné z: https://stackoverflow.com/questions/2480650/role-of-bias-in-neural-networks

[8] HAKL, František a Martin HOLEŇA. *Úvod do teorie neuronových sítí*. Praha, 1997. ČVUT.

[9] SHARMA, Avinash. *Understanding Activation Functions in Neural Networks*[online]. [cit. 2019-01-26]. Dostupné z: https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0

[10] SHARMA, Sagar. *Activation Functions: Neural Networks* [online]. [cit. 2019-01-26]. Dostupné z: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

[11] Obrázek: BHANDE, Anup. *What is underfitting and overfitting in machine learning and how to deal with it.* [online]. [cit. 2019-01-26]. Dostupné z: https://medium.com/greyatom/what-is-underfitting-and-overfitting-in-machine-learning-and-how-to-deal-with-it-6803a989c76

[12] VENCATESAN, M. *Artificial Intelligence vs. Machine Learning vs. Deep Learning* [online]. [cit. 2019-01-26]. Dostupné z: https://www.datasciencecentral.com/profiles/blogs/artificial-intelligence-vs-machine-learning-vs-deep-learning

[13] MAŠEK, Jan Automatické strojové metody získávání znalostí z multimediálních dat: dizertační práce. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2016. 117 s. Vedoucí práce byl doc. Ing. Radim Burget, Ph.D.

[14] *Definition - What does Embedded device mean?* [online]. [cit. 2019-01-26]. Dostupné z: https://www.techopedia.com/definition/10179/embedded-device

[15] *What's your definition of Embedded - discussion* [online]. [cit. 2019-01-26]. Dostupné z: https://www.embedded.com/electronics-blogs/-include/4026109/What-s-your-definition-of-embedded-

[16] GEBHARDT, Daniel, Keyur PARIKH a Iryna DZIECIUCH. *Convolutional Neural Network on Embedded Linux® System-on-Chip* [online]. [cit. 2019-01-26]. Dostupné z: https://apps.dtic.mil/dtic/tr/fulltext/u2/1009093.pdf

[17] Obrázek: Konvoluce. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2019-01-26]. Dostupné z: https://cs.wikipedia.org/wiki/Konvoluce

[18] KARPATHY, Andrej. *Convolutional Neural Networks for Visual* [online]. [cit. 2019-01-26]. Dostupné z: http://cs231n.github.io/convolutional-networks/#case

[19] Obrázek: PATEL, Shyamal a Johanna PINGEL. *Video Introduction to Deep Learning: What Are Convolutional Neural Networks?* [online]. [cit. 2019-01-26]. Dostupné z: https://www.mathworks.com/videos/introduction-to-deep-learning-what-are-convolutional-neural-networks--1489512765771.html

[20] GANDHI, Rohith. *R-CNN, Fast R-CNN, Faster R-CNN, YOLO — Object Detection Algorithms* [online]. [cit. 2019-01-26]. Dostupné z: https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e

[21] UIJLINGS, J.R.R., K.E.A. van de SANDE, T. GEVERS a A.W.M. SMEULDERS. *Selective Search for Object Recognition* [online]. [cit. 2019-01-26]. Dostupné z: http://www.huppelen.nl/publications/selectiveSearchDraft.pdf

[22] TORREY, Lisa a Jude SHAVLIK. *Transfer Learning* [online]. [cit. 2019-01-26]. Dostupné z: http://ftp.cs.wisc.edu/machine-learning/shavlik-group/torrey.handbook09.pdf

[23] *A Beginner's Guide to Restricted Boltzmann Machines* [online]. [cit. 2019-01-26]. Dostupné z: https://skymind.ai/wiki/restricted-boltzmann-machine

[24] *Advances in Neural Networks - ISNN 2017*. Hokkaido, Japan, 2017 [cit. 2019-01-26]. ISBN 978-3-319-59081-3.

[25] CHANDRAYAN, Pramod. *Deep Learning : Deep Belief Network Fundamentals*[online]. [cit. 2019-01-26]. Dostupné z: https://codeburst.io/deep-learning-deep-belief-network-fundamentals-d0dcfd80d7d4

[26] Obrázek: TOPCU, Hasan H. *Deep Belief Networks* [online]. [cit. 2019-01-26]. Dostupné z: https://www.slideshare.net/HasanHTopcu/deep-belief-networks-58155447

[27] *Stacked Autoencoders* [online]. [cit. 2019-01-26]. Dostupné z: http://ufldl.stanford.edu/wiki/index.php/Stacked_Autoencoders

[28] Obrázek: *Stacked Auto-Encoder* [online]. [cit. 2019-01-26]. Dostupné z: https://wikidocs.net/3413

[29] GHODSI, Zahra, Tianyu GU a Siddharth GARG. *SafetyNets: Verifiable Execution of Deep Neural Networks on an Untrusted Cloud* [online]. [cit. 2019-01-26]. Dostupné z: https://papers.nips.cc/paper/7053-safetynets-verifiable-execution-of-deep-neural-networks-on-an-untrusted-cloud.pdf

[30] KHAIRO, Mazin Omar. *The Use of Neural Networks in the Cloud Computing Environment* [online]. [cit. 2019-01-26]. Dostupné z: https://www.iiste.org/Journals/index.php/JIEA/article/viewFile/35438/36459

[31] MACHO, Tomáš. *Mikroprocesory*. Brno, 2017. Skriptum. VUT.

[32] KHAN, Javeed Ahmed, Salalah OMAN, S. RAVICHANDRAN a K. GOPALAKRISHNAN. *Cellular Neural Network on Digital Signal Processor: An Algorithm for Object Recognition* [online]. [cit. 2019-01-26]. Dostupné z: https://www.tandfonline.com/doi/full/10.1080/15325001003652892?

[33] YOSHIDA, MATSUDA, SATO a SUZUMURA. *An artificial neural network accelerator using general purpose 24 bit floating point digital signal processors* [cit. 2019-01-26].

[34] VANHOUCKE, Vincent, Andrew SENIOR a Mark Z. MAO. *Improving the speed of neural networks on CPUs* [online]. [cit. 2019-01-26]. Dostupné z: https://ai.google/research/pubs/pub37631

[35] Obrázek: CULURCIELLO, Eugenio. *Analysis of deep neural networks* [online]. [cit. 2019-01-26]. Dostupné z: https://medium.com/@culurciello/analysis-of-deep-neural-networks-dcf398e71aae

[36] HAO, Yufeng. *A General Neural Network Hardware Architecture on FPGA* [online]. [cit. 2019-01-26]. Dostupné z: https://arxiv.org/ftp/arxiv/papers/1711/1711.05860.pdf

[37] NALLATECH. *FPGA Acceleration of Binary Neural Networks* [online]. [cit. 2019-01-26]. Dostupné z: https://www.nallatech.com/fpga-acceleration-binary-neural-networks/

[38] INTEL. *Efficient Implementation of Neural Network Systems Built on FPGAs, and Programmed with OpenCLTM* [online]. [cit. 2019-01-26]. Dostupné z:

https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/solution-sheets/efficient_neural_networks.pdf

[39] *In-Datacenter Performance Analysis of a Tensor Processing Unit TM: 44th International Symposium on Computer Architecture* [online]. [cit. 2019-01-26]. Dostupné z: https://arxiv.org/ftp/arxiv/papers/1704/1704.04760.pdf

[40] *What makes TPUs fine-tuned for deep learning?* [online]. [cit. 2019-01-26]. Dostupné z: https://cloud.google.com/blog/products/ai-machine-learning/what-makes-tpus-fine-tuned-for-deep-learning

[41] OSBORNE, Joe. *Google's Tensor Processing Unit explained: this is what the future of computing looks like* [online]. [cit. 2019-01-26]. Dostupné z: https://www.techradar.com/news/computing-components/processors/google-s-tensor-processing-unit-explained-this-is-what-the-future-of-computing-looks-like-1326915

[42] KENNEDY, Patrick. *Case Study on the Google TPU and GDDR5 from Hot Chips 29*[online]. [cit. 2019-01-26]. Dostupné z: https://www.servethehome.com/case-study-google-tpu-gddr5-hot-chips-29/

[43] SATO, Kaz, Cliff YOUNG a David PATTERSON. *An in-depth look at Google's first Tensor Processing Unit (TPU)* [online]. [cit. 2019-01-26]. Dostupné z: https://cloud.google.com/blog/products/gcp/an-in-depth-look-at-googles-first-tensor-processing-unit-tpu

[44] Obrázek: CONG, Jason. *Machine Learning on FPGA's* [online]. [cit. 2019-01-26]. Dostupné z: http://cadlab.cs.ucla.edu/~cong/slides/HALO15_keynote.pdf

[45] SAPUNOV, Grigory. *Hardware for Deep Learning. Part 3: GPU* [online]. [cit. 2019-01-26]. Dostupné z: https://blog.inten.to/hardware-for-deep-learning-part-3-gpu-8906c1644664

[46] *Jetson AGX Xavier Developer Kit* [online]. [cit. 2019-01-26]. Dostupné z: https://developer.nvidia.com/embedded/buy/jetson-agx-xavier-devkit

[47] *AlphaGo* [online]. [cit. 2019-02-21]. Dostupné z: https://deepmind.com/research/alphago/

[48] *Neural Compute Stick* [online]. 12. 9. 2018 [cit. 2019-03-07]. Dostupné z: https://software.intel.com/en-us/neural-compute-stick

[49] KRISHEVSKY, Alex, Ilya SUTSKEVER a Hinton GEOFFREY E. *ImageNet Classification with Deep Convolutional Neural Networks*[online]. 6. 6. 2017 [cit. 2019-03-07]. DOI: 10.1145/3065386.

[50] SZEGEDY, Christian, Wei LIU, Yangqing JIA a kolektiv. *Going Deeper with Convolutions* [online]. 7. 6. 2015 [cit. 2019-03-09]. DOI: 10.1109/CVPR.2015.7298594. Dostupné z: https://ieeexplore.ieee.org/document/7298594

[51] HARVEY, Matt. *Creating insanely fast image classifiers with MobileNet in TensorFlow* [online]. [cit. 2019-03-10]. Dostupné z: https://hackernoon.com/creating-insanely-fast-image-classifiers-with-mobilenet-in-tensorflow-f030ce0a2991

[52] HOWARD, Andrew G., Menglong ZHU, Bo CHEN a kolektiv. *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications* [online]. [cit. 2019-03-10]. Dostupné z: https://arxiv.org/pdf/1704.04861.pdf

[53] HE, Kaiming, Xiangyu ZHANG, Shaoqing REN a kolektiv. *Deep Residual Learning for Image Recognition* [online]. 27. 6. 2016 [cit. 2019-03-10]. DOI: 10.1109/CVPR.2016.90.

[54] ZENG, Guozhao, Xiao HU a Yueyue CHEN. *Optimizing Convolution Neural Network on the TI C6678 multicore DSP* [online]. [cit. 2019-03-14]. DOI: 10.1051/matecconf/201824603044. Dostupné z: https://www.matec-conferences.org/articles/matecconf/pdf/2018/105/matecconf_iswso2018_03044.pdf

[55] IGNATOV, Andrey, Radu TIMOFTE, William CHOU a kolektiv. *AI Benchmark: Running Deep Neural Networks on Android Smartphones* [online]. [cit. 2019-03-14]. DOI: 10.1007/978-3-030-11021-5_19. Dostupné z: https://arxiv.org/pdf/1810.01109.pdf

[56] REESE, Lynnette. *Comparing Hardware for Artificial Intelligence: FPGAs vs. GPUs vs. ASICs* [online]. [cit. 2019-03-28]. Dostupné z: http://eecatalog.com/intel/2018/07/24/comparing-hardware-for-artificial-intelligence-fpgas-vs-gpus-vs-asics/

[57] NURVITADHI, Eriko, David SHEFFIELD, Jaewoong SIM a kolektiv. *Accelerating Binarized Neural Networks: Comparison of FPGA, CPU, GPU, and ASIC* [online]. [cit. 2019-03-28]. DOI: 10.1109/FPT.2016.7929192. Dostupné z: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7929192

[58] FRANKLIN, Dustin. *Jetson NANO AI computing to everyone* [online]. [cit. 2019-04-22]. Dostupné z: https://devblogs.nvidia.com/jetson-nano-ai-computing/

[59] *Performance Ranking* [online]. [cit. 2019-04-22]. Dostupné z: http://ai-benchmark.com/ranking

[60] GAUEN, Kent, Ryan DAILEY, John LAIMAN a kolektiv. *Comparison of Visual Datasets for Machine Learning* [online]. [cit. 2019-04-29]. DOI: 10.1109/IRI.2017.59. Dostupné z: https://users.cs.fiu.edu/~chens/PDF/IRI17_Gauen.pdf

[61] *Goodfellow I., Bengio Y., Courville A.: Deep Learning. MIT Press, 2016. ISBN 9780262035613. (deeplearningbook.org)*

[62] TSANK, Sik-Ho. *Review: SSD — Single Shot Detector (Object Detection)* [online]. [cit. 2019-05-05]. Dostupné z: https://towardsdatascience.com/review-ssd-single-shot-detector-object-detection-851a94607d11

[63] KANEDA, Takashi. *MobileNet V2 caffe implementation for NVIDIA DIGITS* [online]. [cit. 2019-05-05]. Dostupné z: https://gist.github.com/kndt84/6492cc0a082245e99a46c0ae8ff25492

[64] *MobileNet-Caffe* [online]. [cit. 2019-05-05]. Dostupné z: https://github.com/shicai/MobileNet-Caffe/blob/master/mobilenet_v2.caffemodel

[65] *Tensorflow/models* [online]. [cit. 2019-05-05]. Dostupné z: https://github.com/tensorflow/models/tree/master/research/object_detection

Seznam symbolů, veličin a zkratek

DL	-	Hluboké učení (z anglického Deep Learning)
ML	-	Strojové učení (z anglického Machine Learning)
CNN	-	Konvoluční neuronová síť (z angl. Convolutional Neural Network)
NN	-	Neuronová síť (z angl. Neural Network)
DSP	-	Digitální signálový procesor
FPGA	-	Programovatelné pole logických hradel
API	-	Programové programovací rozhraní
GUI	-	Grafické uživatelské rozhraní
CPU	-	Centrální procesorová jednotka
GPU	-	Grafická jednotka
ASIC	-	Integrovaný chip, navržený pro jednu aplikaci
L4T	-	Linux 4 Tegra, verze operačního systému Linux

Seznam příloh

Příloha 1. CD/DVD
Příloha 2. Obsah přiloženého disku

PŘÍLOHA 2

/
README.txt Tento soubor obsahuje bližší informace o obsahu CD
Skripty/
……Create_TensorRT_model.py
……Jetson_camera_object_detection.py
……Jetson_video_object_detection.py
……PC_image_object_detection.py
……PC_video_object_detection.py
……Pi_image_object_detection.py
……Pi_video_object_detection.py
Modely/
……..DIGITSMobileNetV2.rar
……..MaoamInceptionV2.rar
……..MaoamMobileNetV2.rar
Maoam_dataset/
……..Kitti_format.rar
……..PascalVOC_format.rar